Schedulability Analysis of Distributed Multi-core Avionics Systems with UPPAAL

Han, Pujie; Zhai, Zhengjun; Nielsen, Brian; Nyman, Ulrik; Kristjansen, Martin

Published in:
Journal of Aerospace Information Systems

DOI (link to publication from Publisher):
10.2514/1.I010715

Publication date:
2019

Document Version
Accepted author manuscript, peer reviewed version

Schedulability Analysis of Distributed Multi-core Avionics Systems with UPPAAL

Pujie Han* and Zhengjun Zhai†
Northwestern Polytechnical University, Xi'an, 710072, China

Brian Nielsen‡, Ulrik Nyman§, and Martin Kristjansen¶
Aalborg University, Aalborg, 9220, Denmark

This paper presents an approach for schedulability analysis of Distributed Integrated Modular Avionics (DIMA) systems that consist of spatially distributed ARINC-653 multi-core modules connected by a unified Avionics Full Duplex Switched Ethernet (AFDX) network. We model a multi-core DIMA system as a set of stopwatch automata in UPPAAL to verify its schedulability by model checking. However, direct verification is infeasible due to the large state space. Therefore, we combine global analysis based on Statistical Model Checking (SMC) and compositional analysis based on classical model checking, thereby mitigating the state space explosion problem. Even though the nature of SMC testing cannot prove schedulability, the model of a DIMA system first undergoes quick schedulability falsification using global SMC analysis. Thereafter, we use a compositional approach to check each partition including its communication environment individually. By using assume-guarantee reasoning, we ensure that each real-time task meets the deadline and that communication constraints are also fulfilled globally. The approach is finally applied to the schedulability analysis of a concrete multi-core DIMA system.

I. Introduction

As the cost of avionics systems rapidly increases in the aviation industry, there is a growing trend towards providing a more generalized airborne computation environment for Commercial Off-The-Shelf (COTS) products. The architecture of Distributed Integrated Modular Avionics (DIMA) [1] is proposed for this purpose. It installs standardized computer modules in spatially distributed locations [2] that are connected by a unified deterministic bus system [3] such as an Avionics Full Duplex Switched Ethernet (AFDX) network [4]. To keep a balance between the performance and the Size, Weight and Power (SWaP) consumption of avionics systems, COTS multi-core processors have been

*Ph.D. Candidate, School of Computer Science and Engineering, West Youyi Road 127.
†Professor, School of Computer Science and Engineering, West Youyi Road 127.
‡Associate Professor, Department of Computer Science, Selma Lagerløfs Vej 300.
§Associate Professor, Department of Computer Science, Selma Lagerløfs Vej 300.
¶Ph.D. Student, Department of Computer Science, Selma Lagerløfs Vej 300.

widely applied to airborne computer modules, where avionics applications run in ARINC-653 [5] partitioned operating systems.

Enabled by the continued improvement of semiconductor technology (Moore's law), the current approach to increasing the processor performance at low cost is mainly through the integration of multiple cores in a single processor. This direction is driven by the so-called power-wall that prevents processors from being clocked at an increased rate, and by the very high complexity of designing processors that continues to improve performance through ever increasing levels of implicit instruction level parallelism.

However, multi-core processors also create new challenges at different levels. Applications must be designed with explicit thread level parallelism, often leading to massively concurrent applications, which again makes it more challenging to ensure correct behavior in terms of functionality, timing, and absence of concurrency bugs like race-conditions and deadlocks. For safety- and time-critical avionics system, this specifically increase the challenge of creating and mapping many application partitions (possibly with non-trivial inter- and intra-partition dependencies) on multiple cores (or even networked processors), and being able to perform the required schedulability analysis to guarantee correct timing. At the operating system level, challenges are providing new effective mechanisms for predictable management, scheduling, synchronization, and partitioning of the underlying processing and memory resources, such that schedulability and worst-case execution time analysis are feasible. At a hardware level, challenges concern creating core-interconnects and memory bus systems to allow sufficiently fast data- and instruction transfer to the multiple cores, and providing memory coherency across them and the different memory types found in multi-core systems. As a further consequence, the development cost of safety critical systems is increased by the need for certifying the safe operation of the system.

Multi-core hardware and distributed applications thus lead to increasingly complex systems whose schedulability has been becoming difficult to validate. This paper addresses this challenge by proposing a new method and tool for analyzing the schedulability of complex avionics systems.

~~A schedulable DIMA system should fulfil not only the temporal requirements of each real-time task on multiple processor cores but also communication constraints among the distributed nodes.~~ The development of model checking based approaches has currently become an attractive topic for the schedulability analysis of complex real-time systems due to the sufficient expressiveness of formal models. The techniques of classical model checking (MC) describe schedulability as temporal logic properties and verify the properties via state space exploration. ~~There have been works using model-checking to analyze the temporal behavior of individual avionics modules in various formal models such as Coloured Petri Nets (CPN) [6], preemptive Time Petri Nets (pTPN) [7], Linear Hybrid Automata (LHA) [8], Timed Automata (TA) [9], and StopWatch Automata (SWA) [10].~~ Unfortunately, when being applied to a complete avionics system, they suffer from an inevitable problem of state space explosion, which makes the exact model checking practically infeasible.

Compositional approaches are widely adopted to alleviate the state space explosion problem. Some studies [11–13] exploit the inherent isolation of temporal partitioning by analyzing each partition separately and concluding system properties at a global level, but they ignore the behavior of the underlying network or the interactions among partitions. Thus these methods are not applicable to DIMA environments in which multiple distributed ARINC-653 partitions communicate through a shared network to perform an avionics function together.

Statistical Model Checking (SMC) [14] is also proposed as a promising technique that has the powerful facilities of formal modeling as well as avoids the state space explosion of classical model checking. An SMC engine runs and monitors a number of simulation processes in order to quickly estimate the statistical results of the satisfaction or violation of certain properties. However, the SMC cannot provide any guarantee of schedulability but quick falsification owing to its nature of statistical testing. Therefore, it is reasonable to apply both classical and statistical model checking to the schedulability analysis of avionics systems.

In this paper, we present an approach to schedulability analysis of multi-core DIMA systems that are modeled as a set of StopWatch Automata (SWA) in UPPAAL. The approach combines compositional and global analysis by classical and statistical model checking. The paper is a combination and extension of the two workshop papers [15] and [16]. The rest of the paper is organized as follows. Section II presents related work and contributions. Section III describes the structure of DIMA systems, providing the modeling requirements for a DIMA system. The UPPAAL models and their schedulability analysis are presented in section IV. In section V we detail the compositional analysis approach. In section VI we present a case study and its experimental results, and section VII discusses those same results along with their applicability to related schedulability problems. Finally, section VIII concludes the paper.

II. Related Work and Contributions

The structure of DIMA systems has been fully discussed in [1–3, 17]. The current research into the schedulability of DIMA systems focuses on two of their major constituent parts: ARINC-653 modules and the underlying AFDX network.

A multitude of analytical methods [18–23] can be used to analyze the schedulability of ARINC-653 modules, which belong to two-level hierarchical scheduling systems. However, the worst-case assumptions in these analytical methods are more pessimistic than real situations. By contrast, ~~the~~ model checking based approaches are more expressive and can be used to perform exact schedulability analyses on the basis of various formal models such as Coloured Petri Nets (CPN) [6], preemptive Time Petri Nets (pTPN) [7], Linear Hybrid Automata (LHA) [8], Timed Automata (TA) [9], and Stopwatch Automata [10].

The authors of [6] construct a CPN model to describe real-time task scheduling in a generic avionics mission computer, deriving task response times from the model to determine the feasibility of five different scheduling protocols. Nevertheless, the modeling method does not cover any ARINC-653 features and is only applicable to

federated avionics. In [24], the theory of pTPN is introduced to support exact schedulability analysis of two-level hierarchical scheduling systems. This approach is extended in [11] with a concept of required interface that models the environment of each application to enable compositional analysis of hierarchical scheduling systems encompassing inter-application communications between periodic tasks. However, the communication latency of the underlying bus system and the features of ARINC-653 ports are not taken into account. The authors of [12] employ the formalism of LHA to model a hierarchical scheduling system where there is no communication among tasks. They adopt a dynamic server algorithm, separating each server from the rest of the system, and thereby enabling component-based scheduling analysis. This method is subsequently applied to a multi-core global scheduling system and extended with a weak simulation in [25] to reduce the state space of complex models. The approaches of [26, 27] use TA to describe a hierarchical scheduling system, while both of them are limited to periodic tasks isolated from each other. The formal modeling of pTPN and the verification of SWA are combined in [28], which allows for inter-task communication but no temporal partitioning mechanism. The authors of [13] introduce their compositional frameworks using SWA for analyzing the schedulability of hierarchical scheduling systems. The modeling is covering concrete task actions and intra-partition synchronization and therefore allows for more features of avionics systems, but the compositional analysis does not support the communication among partitions. They also apply a similar modeling method to the evaluation of multi-core platforms in [29, 30]. Obviously, all of the aforementioned approaches lack the capability to model and analyze network communication in DIMA systems from a global viewpoint.

Up to now, several approaches have been introduced for calculating end-to-end delays in an AFDX network, including a simulation approach [31, 32], response time analysis [33, 34], network calculus [35, 36], trajectory approach [37], forward end-to-end delays analysis [38, 39] and model checking approach [40–42]. Although the model checking approach can obtain more exact results than the rest that compute the upper bounds of worst-case end-to-end delays, it is confronted with the state space explosion problem for realistic networks [41].

The objective of this paper is to help apply model checking to the schedulability analysis of a multi-core DIMA system. From a global perspective of DIMA systems, we employ the combination of SMC and a compositional approach to cope with the state space explosion problem of classical model checking. The main contributions of this paper are summarized as follows:

- *Comprehensive Modeling of multi-core DIMA systems* that covers the features of two-level ARINC-653 compliant multi-core schedulers, periodic/sporadic tasks, intra-partition synchronization, and inter-partition communications through an AFDX network.
- *A method for global analysis using statistical model checking* allows users to quickly falsify non-schedulable configurations by SMC hypothesis testing, which can handle a complete system model and avoid an exhaustive exploration of the state space.
- *A method for compositional analysis using classical model checking* verifies the model of each ARINC-653

partition including its environment individually and then assembles the local results together to derive conclusions about the schedulability of an entire system. A compositional approach performs assume-guarantee reasoning [43] to reduce the complexity of symbolic model-checking.

- *An abstraction relation*, timed selection simulation relation, which allows users to create a set of abstract models that collectively describe the external behavior of a concrete model, thereby simplifying the abstraction in assume-guarantee reasoning.
- A notion of *message interfaces* that decouples the communication dependencies between partitions. By composing any partition with its related message interfaces and verifying safety properties of the composition, we can conclude that these properties are still preserved at the global level.
- Application of the approach to an avionics case study, thus validating the feasibility of the approach.

This paper is a combination and extension of the two workshop papers [15] and [16]. These previous papers each focused on one aspect of the approach, namely the models [15] and compositionality [16].

On top of combining the two papers this journal paper also adds several completely new elements and joins it all to a coherent approach:

- Multi-core aspects
- Section III with added description of the ARINC-653 multi-core modes
- Experiments using a symmetric multi-processor scheduler
- Detailed description of all UPPAAL models
- New AFDX models that are able to describe a network topology, giving a more precise latency analysis for the communication between nodes in the network

III. Avionics System Description

This section presents the structure and the main parts of a DIMA system. Thereafter, the two types of ARINC-653 multi-core systems are presented and described. Based on the structure of such systems and the multi-core aspects, formal terms are defined in order to use the properties of a DIMA system in a schedulability analysis.

A. Distributed Integrated Modular Avionics System

In DIMA systems, a unified AFDX network connects standardized computer devices and thousands of peripherals (sensors and actuators), which are linked by Remote Data Concentrators (RDC) to the AFDX network [3]. Fig. 1 shows a simplified example of a DIMA system, where physically distributed DIMA core modules [5] execute application tasks simultaneously to fulfill avionics functions cooperatively.

In the aviation industry, ARINC-653 series standards define a general-purpose APEX (APplication/EXecutive) interface between the operating system and the application software, providing a space and time partitioning mechanism

for avionics applications [5]. Thus the tasks resident on each DIMA core module run in an ARINC-653 partitioned operating system which realizes a two-level scheduling mechanism and achieves temporal isolation between ARINC-653 partitions. In such a scheduling system, partitions are scheduled by a Time Division Multiplexing (TDM) global scheduler and each partition also has its local scheduling policy based on preemptive Fixed Priority (FP) to manage the internal tasks [5].

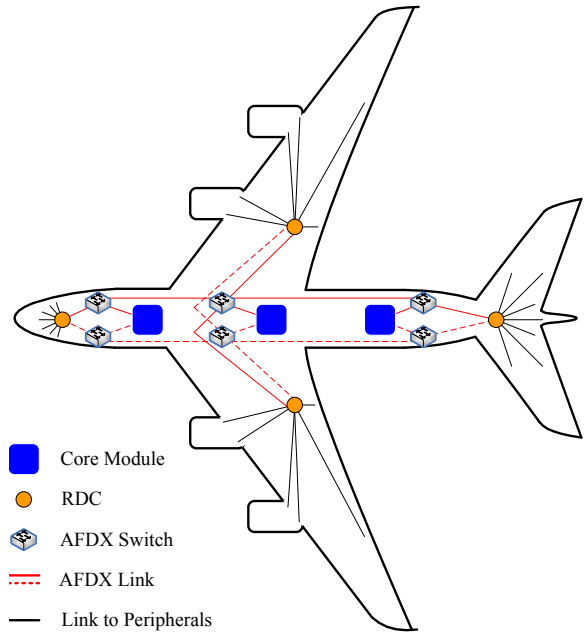


Fig. 1 An example of a DIMA system

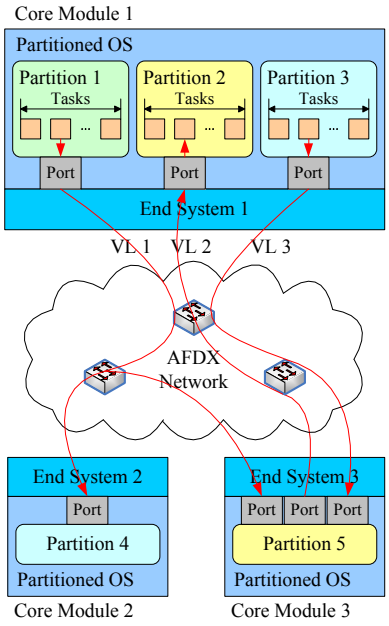


Fig. 2 A DIMA core system

The distributed nature of DIMA systems lead to frequent inter-partition communication not only within core modules but also between them through the underlying AFDX network. According to the ARINC-653 standard [5], all inter-partition communication is conducted using a message passing system. Messages originating from partitions are only allowed to be communicated via an ARINC-653 port and transmitted through a logical channel from a single source partition to one or more destination partitions. Each port and its logical channel are associated with one of the modes of transfer: sampling mode or queuing mode. A sampling port can accommodate at most a single message that remains in the buffer until it is either transmitted by the channel from a source port or overwritten by a new message in a source or destination port. Moreover, a refresh period is defined as an attribute of each sampling port. This attribute provides a correct arrival rate to determine the validity of received messages, regardless of the rate of receiving requests from application tasks. In contrast, a queuing port, which only supports unicast connections, is allowed to buffer multiple messages in a message queue with a fixed capacity. However, the operating system is not responsible for handling overflow after the message queue is full. We consider three schedulability properties that a system has to satisfy: task deadlines, refresh periods of sampling ports and non-overflow of queuing ports. These properties are detailed in Section IV.B.

In this paper, we consider the DIMA core system as shown in Fig. 2 where an AFDX network connects all the core modules through their End Systems (ES), each of which provides a hardware implementation of the AFDX protocol stack [4]. ARINC-664 part 7 [4] prescribes that an AFDX ES should provide ARINC-653 port services for the operating system. A message sent from a source port is expanded by the UDP and IP header at the UDP/IP layer and then forwarded to a Virtual Link (VL), which defines a logical connection from one source ES to one or more destination ESs. Unlike the Internet, routing in AFDX network is based on VLs instead of IP address. What is more, ESs provide each VL with a dedicated ~~maximum~~ bandwidth to ensure an upper bound on end-to-end delay. The message flows belonging to the same VL share the bandwidth. In transmitting between ESs, a scheduler regulates and multiplexes the flows from different VLs. This single multiplex flow is sent across two independent switched networks and both of them will arrive at the destination ES(s) under normal operation. The underlying network redundancy is transparent to the upper protocols because of the function of redundancy management in each ES. Throughout this paper, we assume that there is no unmasked fault in our AFDX network.

B. Implementation of ARINC-653 Multi-core systems

Although multi-core processors have been widely applied across various application domains, ARINC-653 series standards did not cover the features of multi-core processors until the ARINC-653 Part 1 Supplement 4 (ARINC653P1-4) was published in 2015. The leading software vendors of ARINC-653 compliant systems thus implement multi-core aspects using different strategies. According to the level of parallelism, two types of software architecture are commonly utilized to implement an ARINC-653 compliant system on multi-core platforms: Asymmetrical Multi-Processing (AMP) and Symmetrical Multi-Processing (SMP) [44].

An AMP deployment enables inter-partition parallelism in one processor, where each core is managed by a separate instance of an operating system that provides airborne software application with the ARINC-653 partitioning mechanism [45]. In such a configuration, each partition is executed on a single core in parallel with the partitions running on the other cores, and thus independent partition schedules are assigned to different cores [46]. Since the operation of each partition is similar to that on a single core processor, the AMP architecture especially suits the compatibility requirements for legacy applications.

However, ARINC653P1-4 did not define inter-partition parallelism within an operating system [5]. When applying an AMP architecture, it requires more effort from the platform providers to handle the contention and co-operation of individual operating systems on different cores. As shown in Fig. 3, the practical AMP implementation, such as the VxWorks 653 Multi-core Edition developed by Wind River [47], introduce a hypervisor layer running across all the cores, thereby ensuring robust partitioning on the operating system level and concerted access to shared resources of the processor's multiple cores.

By contrast, an SMP deployment realizes task parallelism inside one partition that is activated on all cores

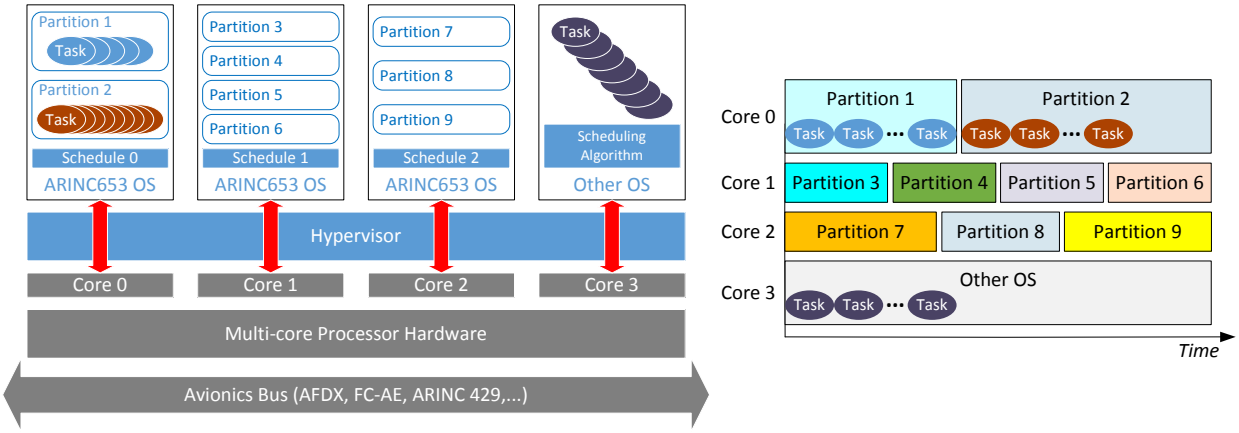


Fig. 3 AMP architecture and its multi-core scheduling examples

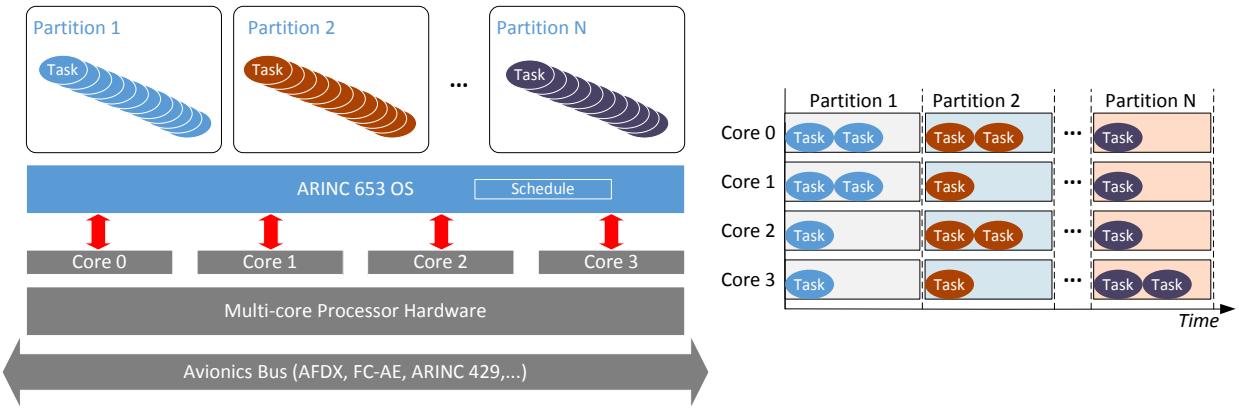


Fig. 4 SMP architecture and its multi-core scheduling examples

synchronously, which means the tasks belonging to a common partition can run in parallel on different cores. In the SMP mode, as depicted in Fig. 4, all the cores are managed by a single instance of an operating system where only one partition schedule is required. The SMP configuration applies to a substantial number of emerging airborne software applications, such as radar, sonar and image processing, due to their inherent parallelism [48]. Nevertheless, the versions earlier than ARINC653P1-4 did not consider such use of multi-core processors. The legacy applications developed under these widely-used versions may depend on sequential execution and fail inside a parallel partition. By adopting an SMP configuration, function suppliers can benefit from the use of multi-core processors to improve the performance of specific airborne software applications, but have to put more effort into parallel programming schedulability analysis.

The SMP architecture is compatible with ARINC653P1-4 in the scope of an operating system. There are two attributes that control the mappings between partitions or tasks and processor cores in ARINC653P1-4. The fixed attribute *Assigned Processor Core(s)* assigns a particular set of processor cores to a partition. Each task is further associated with an attribute *Processor Core Affinity*, which specifies the processor core or cores the task can run on. A

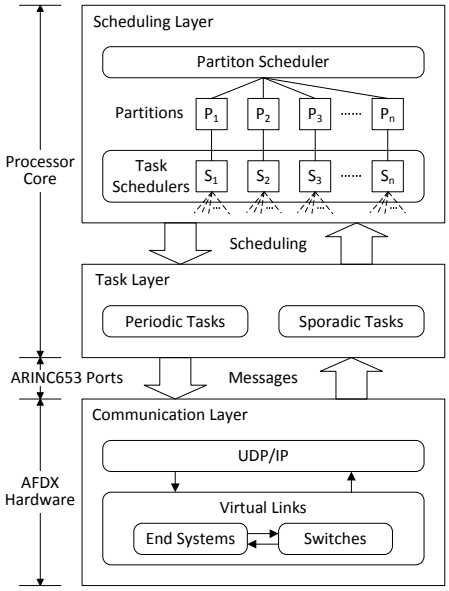


Fig. 5 Layered structure of DIMA core systems

task can only be assigned an affinity for one of the processor cores allocated to the partition to which the task belongs. Therefore, each processor core assigned to the partition will execute a different fixed set of tasks simultaneously [5].

C. Layered Model of an Avionics System

Fig. 5 shows a three-layer model of a DIMA core systems consisting of scheduling layer, task layer, and communication layer. The constituent elements are detailed as follows.

The *scheduling layer* comprises the hierarchical scheduling facilities of an ARINC-653 module. The multi-core architecture AMP and SMP differ in the constitution of this scheduling layer.

- In an AMP configuration, each processor core has an independent TDM *partition scheduler* $\langle Core, MF, Sch\rangle$, where *Core* is an identifier of the processor core, *MF* is a major time frame and *Sch* is a partition schedule. Since partitions are scheduled on a fixed cyclic basis, the partition-scheduling behavior is periodically repeated every *MF* [5]. *Sch* contains a set of partition windows, i.e. time slots, each of which is defined as $\langle P, Off, Dur\rangle$ where *P* identifies a partition, *Off* denotes the offset from the start of *MF* and *Dur* is the expected duration / budget. Partitions are only activated within their corresponding partition windows [5].

  The SMP configuration equips all the processor cores of a module with a common TDM partition scheduler $\langle MF, Sch, Assi\rangle$ where *MF* and *Sch* have the same definitions as those of the AMP partition scheduler. In addition, the set *Assi* indicates the processor core(s) assigned to partitions. Its elements are defined as $\langle P, Cores\rangle$ where *P* identifies a partition and *Cores* is a set of processor cores.
- The AMP configuration assigns each partition a preemptive FP *task scheduler*, which always selects the task with the highest priority in the ready state within the partition to run.

The SMP task scheduler uses a partitioned preemptive FP scheduling policy [49]. When a partition is assigned multiple processor cores, a set of tasks in the ready state may be selected based on both priority and core affinity to run concurrently on the assigned processor cores [5].

All the application tasks executing avionics functions constitute the *task layer*. We consider a task as the smallest scheduling unit, each of which can be executed concurrently with other tasks in the same partition. A task is indicated by the tuple $\langle \mathcal{I}, \mathcal{P}_{min}, \mathcal{P}_{max}, \mathcal{O}, \mathcal{J}, \mathcal{D}, \mathcal{C}, \mathcal{R}, \mathcal{L} \rangle$ where $\mathcal{I}$ is initial offset determining the first release point of the task, $\mathcal{P}_{min}$ and $\mathcal{P}_{max}$ are the minimum period and the maximum period respectively, $\mathcal{O}$ is offset, $\mathcal{J}$ is jitter, $\mathcal{D} \leq \mathcal{P}_{min}$ is the deadline, $\mathcal{C}$ denotes processor core affinity, $\mathcal{R}$ denotes task priority, and $\mathcal{L}$ is a sequential list of abstract instructions. In the task model, jobs of each task are scheduled repeatedly and a task releases the kth job ($k \in N$) in the time interval $[\mathcal{I} + k\mathcal{P}_{min} + \mathcal{O}, \mathcal{I} + k\mathcal{P}_{max} + \mathcal{O} + \mathcal{J}]$. Let t_k be the time when the kth job is released. For any task in a partition P_i, we define the following two task types:

- A *periodic task* has the kth release time $t_k \in [\mathcal{I} + k\mathcal{P} + \mathcal{O}, \mathcal{I} + k\mathcal{P} + \mathcal{O} + \mathcal{J}]$ where $\mathcal{P} = \mathcal{P}_{min} = \mathcal{P}_{max} < +\infty$ is a fixed period.
- A *sporadic task* characterized by a minimum separation $\mathcal{S} = \mathcal{P}_{min} - \mathcal{J}$ between consecutive jobs releases its $(k+1)$th job at $t_{k+1} \in [t_k + \mathcal{S}, +\infty)$, and its first release time t_0 is in the interval $[\mathcal{I} + \mathcal{O}, +\infty)$.

An element of the sequential list $\mathcal{L}$ represents the operation of an abstract instruction $\langle Cmd, Res, T_{BCET}, T_{WCET} \rangle$. *Cmd* is the type of abstract instructions belonging to the command set {*Compute*, *Lock*, *Unlock*, *Delay*, *Send*, *Receive*, *End*}. *Res* is an identifier encoding one of the resources such as CPU, mutual exclusion locks, messages, and ports. T_{BCET} and T_{WCET} are the execution time in the best and the worst case, respectively. In the command set, *Compute* represents a general computation step, *Lock* and *Unlock* handle mutual exclusion locks, *Delay* allows the current task to stop running for a certain time, *Send* and *Receive* are utilized in inter-partition communications, and *End* is the symbol of job termination. Since the direct application of mutual exclusion locks can cause an unpredictable duration of priority inversion, we adopt the priority ceiling protocol [50] to deal with intra-partition synchronization.

The *communication layer* interacts with the task layer through ARINC-653 ports and provides the services of inter-partition communication between different modules. We focus on the transfer latency in the communication layer. According to the structure of the AFDX protocol stack, the communication layer is further divided into an UDP/IP layer and virtual links.

- The latency of delivery through the *UDP/IP layer* in each ES is commonly implementation dependent but can be measured and bounded on an interval $[Tu_{min}, Tu_{max}]$, which indicates the delay in forwarding a message from an ARINC-653 port (Transmission) or an Ethernet frame from a VL (Reception) to its destination buffer.
- A *virtual link* is characterized as the tuple $\langle L_{max}, BAG, N, Conn \rangle$ where L_{max} is the maximum frame length and BAG is the bandwidth allocation gap, i.e. the minimum interval between the first bits of two consecutive frames [4]. Thus, the maximum available bandwidth of the VL is L_{max}/BAG. N denotes the speed of the

physical link. Given a frame length L in bytes, the frame delay is $(8 \times L)/N$, i.e. the time taken to deliver the frame to a physical link [4]. Let H and W be the set of ESs and switches respectively. $Conn \subseteq (H \cup W) \times (H \cup W)$ defines the set of VL's physical links, representing a logical connection from one source ES to one or more destination ESs. The ESs in a VL are connected by switches. In order to model the latency through a VL, we consider the accumulated value of the technological latency (independent of traffic load) and configuration latency (depending on configuration and traffic load) [4] along the logical path.

Additionally, two *message patterns* are provided for inter-partition messages between the task and communication layer. Since message-sending actions and the release of their source tasks have similar temporal features, we define two patterns of periodic and sporadic messages that are generated by periodic and sporadic tasks respectively. For any message type msg_k, we associate a time stamp $t_l^k, l \in \mathbf{N}$, which is the accumulated time since the initial instant, with each message-sending action. Time stamps t_l^k identify two message patterns:

- *periodic messages* with the time stamps $t_l^k \in [\mathcal{I}^k + l\mathcal{P}^k + \mathcal{O}^k, \mathcal{I}^k + l\mathcal{P}^k + \mathcal{O}^k + \mathcal{J}^k]$, and
- *sporadic messages* having the $(l+1)$th time stamp $t_{l+1}^k \in [t_l^k + \mathcal{P}^k - \mathcal{J}^k, +\infty)$ after its first time stamp $t_0^k \in [\mathcal{I}^k + \mathcal{O}^k, +\infty)$,

where $\mathcal{I}^k$ is initial offset, $\mathcal{P}^k$ is period, $\mathcal{O}^k$ is offset, and $\mathcal{J}^k$ is jitter. By instantiating the parameters of a pattern, one can describe the message-sending behavior of a specific task.

The above coupled elements represent the parallel components of a DIMA core system, which can be modeled as a network of SWA. In the following sections, we use Ω to denote the set of SWA in system models.

IV. Modeling Framework

This sections is divided into four parts. The necessary preliminaries for this paper are presented followed by an overview of the modelling framework. The framework presents what kind of TA templates are needed in order to represent the behaviour of a DIMA system. Thereafter, the model-based analysis method is presented. Here it is described how classical MC and SMC are combined in order to mitigate state space explosion. Lastly, the TA templates in Uppaal are presented in detail.

A. Preliminaries

We present formal definitions including SWA with an input/output extension and its semantic object Timed I/O Transition Systems (TIOTSs) [51].

Suppose that C is a finite set of clocks and V is a finite set of integer variables. A *valuation* $u(x)$ with $x \in C \cup V$ denotes a mapping from C to $\mathbf{R}_{\geq 0}$ and from V to $\mathbf{N}$. Let $LC(C, V)$ be the set of linear constraints. A *guard* $g \in LC(C, V)$ is a linear constraint which is defined as a finite conjunction of atomic formulae in the form of $c \sim n$, $c - c' \sim n$ or $v \sim n$ with $c, c' \in C, v \in V, n \in \mathbf{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$. Given any valuation u, we change the values of clocks and

integer variables using an *update* operation $r(u) \in 2^R$ in the form of $c = 0$ or $v = n$ where $c \in C, v \in V$ and $n \in \mathbf{N}$, and R is the universal set of update operations. In addition, we define an *action* set Σ used for synchronization, an internal action $\tau \notin \Sigma$, and their universal set $\Sigma^\tau = \Sigma \cup \{\tau\}$.

Definition 1 (Stopwatch Automaton [10]) *A stopwatch automaton is a tuple $\langle Loc, l_0, C, V, E, \Sigma, Inv, drv\rangle$ where Loc is a finite set of locations, $l_0 \in Loc$ is the initial location, C is a finite set of clocks, V is a finite set of integer variables, $E \subseteq Loc \times LC(C,V) \times \Sigma^\tau \times 2^R \times Loc$ is a set of edges, $\Sigma = I \oplus O, I \cap O = \varnothing$ is a finite set of actions divided into inputs(I) and outputs(O), Inv is a mapping $Loc \to LC(C,V)$, and drv is a mapping $Loc \times C \to \{0,1\}$.*

From a syntactic viewpoint, SWA belongs to the class of TA extended with drv, which can prevent part of the clocks called stopwatches from changing in specified locations ~~semantically~~. We now shift the focus to the semantic object TIOTS of SWA.

In a TIOTS, there are two types of transitions: delay and action transitions. We use the set $D = \{\epsilon(d) | d \in \mathbf{R}_{\geq 0}\}$ to denote the delay, and refer to the 0-delay $\epsilon(0)$ as $\mathbf{0}$.

Definition 2 (Timed I/O Transition System) *A timed I/O transition system is a tuple $\mathcal{T} = \langle S, s_0, \Sigma, \to\rangle$ where S is an infinite set of states, s_0 is the initial state, $\Sigma = I \oplus O, I \cap O = \varnothing$ is a finite set of actions divided into inputs(I) and outputs(O), and $\to \subseteq S \times \Sigma^\tau \cup D \times S$ is a transition relation. $s \xrightarrow{a} s'$ represents $(s, a, s') \in \to$, which has the properties of time determinism, time reflexivity, and time additivity [51].*

For any SWA, a state is defined as a pair $\langle l, u\rangle$ where l is a location and u is a valuation over clocks and integer variables. On the basis of TIOTSs, the operational semantics of SWA is defined as follows.

Definition 3 (Semantics of SWA) *The operational semantics of a stopwatch automaton $A = \langle Loc, l_0, C, V, E, \Sigma, Inv, drv\rangle$ is a timed I/O transition system $\mathcal{T}^A = \langle S, s_0, \Sigma, \to\rangle$ where S is the set of states of A, $s_0 = \langle l_0, u_0\rangle$ is the initial state of A, Σ is the same set of actions as A, and $\to$ is the transition relation defined by*

- $\langle l, u\rangle \xrightarrow{a} \langle l', u'\rangle$ *iff* $\exists \langle l, g, a, r, l'\rangle \in E \ \ (u \models g \ \wedge \ u' = r(u) \ \wedge \ u' \models Inv(l'))$
- $\langle l, u\rangle \xrightarrow{\epsilon(d)} \langle l', u'\rangle$ *iff* $l = l' \wedge (\forall v \in V \ \ u'(v) = u(v)) \ \wedge \ (\forall c \in C \ (drv(l,c) = 0 \Rightarrow u'(c) = u(c))) \ \wedge \ (\forall c \in C \ (drv(l,c) = 1 \Rightarrow u'(c) = u(c) + d)) \ \wedge \ u' \models Inv(l')$.

For any transition $s \xrightarrow{a} s'$, two symbols $a?$ and $a!$ denote the action a belonging to input I and output O respectively. Given $a \in \Sigma$, $s \xrightarrow{a}$ iff $\exists s' \in S$, s.t. $s \xrightarrow{a} s'$. $\xrightarrow{\tau}{}^*$ or $\xRightarrow{\mathbf{0}}$ denotes the reflexive and transitive closure of $\xrightarrow{\tau}$. $s \xRightarrow{\epsilon(d)} s'$ iff $s \xrightarrow{\epsilon(d)} s'$, or $\exists s_1, s_2, \ldots, s_n \in S$, s.t. $s \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{n-1}} s_n \xrightarrow{\alpha_n} s'$ and $\forall i \in \{0, \ldots, n\}$, s.t. $\alpha_i = \tau$ or $\alpha_i \in D$ and $d = \sum\{d_i | \alpha_i = \epsilon(d_i)\}$.

The definition of parallel composition $\|$ of TIOTSs is similar to that in [51]. Given two TIOTSs $\mathcal{T}_i = \langle S_i, s_{i,0}, \Sigma_i, \to_i \rangle, i \in \{1, 2\}$, they are *compatible* iff they satisfy the following conditions:

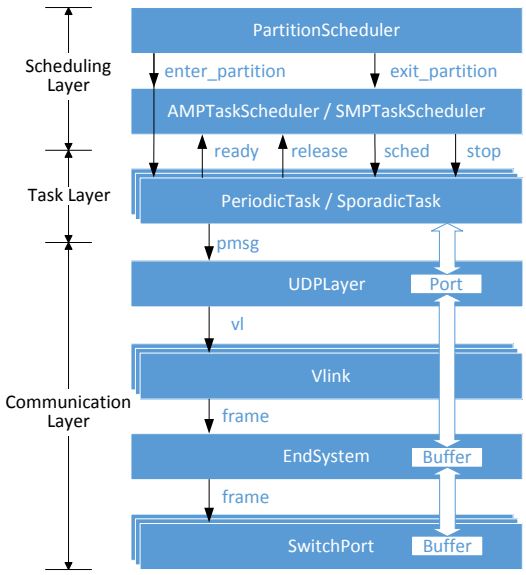


Fig. 6 UPPAAL **modeling framework**

- *(Unique output)* $O_1 \cap O_2 = \varnothing$.
- *(Nonblocking input)* $\forall s \in S_i \ \forall a \in I_i \ s \xrightarrow{a}$.

Note that the nonblocking input actions are realized as broadcast channels in UPPAAL.

Definition 4 (Parallel Composition) *Suppose two timed I/O transition systems* $\mathcal{T}_1 = \langle S_1, s_{1,0}, \Sigma_1, \rightarrow_1 \rangle$ *and* $\mathcal{T}_2 = \langle S_2, s_{2,0}, \Sigma_2, \rightarrow_2 \rangle$ *are compatible. The parallel composition* $\mathcal{T}_1 || \mathcal{T}_2$ *is the timed I/O transition system* $\langle S, s_0, \Sigma, \rightarrow \rangle$ *where* $S = S_1 \times S_2$, $s_0 = \langle s_{1,0}, s_{2,0} \rangle$, $\Sigma = I_{1||2} \oplus O_{1||2}$, $I_{1||2} = (I_1 \setminus O_2) \cup (I_2 \setminus O_1)$, $O_{1||2} = O_1 \cup O_2$, *and* $\rightarrow$ *is the largest relation generated by the following rules:*

- *INDEP-L:* $\dfrac{s_1 \xrightarrow{a} s_1' \quad a \in \{\tau\} \cup \Sigma_1 \setminus \Sigma_2}{\langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1', s_2 \rangle}$ *INDEP-R:* $\dfrac{s_2 \xrightarrow{a} s_2' \quad a \in \{\tau\} \cup \Sigma_2 \setminus \Sigma_1}{\langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1, s_2' \rangle}$
- *DELAY:* $\dfrac{s_1 \xrightarrow{\epsilon(d)} s_1' \quad s_2 \xrightarrow{\epsilon(d)} s_2' \quad d \in \mathbf{R}_{\geq 0}}{\langle s_1, s_2 \rangle \xrightarrow{\epsilon(d)} \langle s_1', s_2' \rangle}$
- *SYNC-IN:* $\dfrac{s_1 \xrightarrow{a} s_1' \quad s_2 \xrightarrow{a} s_2' \quad a \in I_{1||2}}{\langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1', s_2' \rangle}$
- *SYNC-IO:* $\dfrac{s_1 \xrightarrow{a} s_1' \quad s_2 \xrightarrow{a} s_2' \quad a \in (I_1 \cap O_2) \cup (O_1 \cap I_2)}{\langle s_1, s_2 \rangle \xrightarrow{a} \langle s_1', s_2' \rangle}$

For any SWA $A_1, A_2 \in \Omega$, we define the composite model $A = A_1 || A_2$ iff their TIOTSs satisfy $\mathcal{T}^A = \mathcal{T}^{A_1} || \mathcal{T}^{A_2}$.

B. An Overview of the Modeling Framework

The modeling framework is organized as a set of UPPAAL templates with a layered structure. Fig. 6 shows an overview of these templates together with the channels between them.

The scheduling layer consists of three TA templates each responsible for a different scheduling concept: a `PartitionScheduler` and the two task schedulers `AMPTaskScheduler` and `SMPTaskScheduler`. `PartitionScheduler` provides the service of TDM scheduling for partitions. `AMPTaskScheduler` and `SMPTaskScheduler` implement the task scheduling of a particular partition in AMP and SMP multi-core configuration, respectively. The model of a task scheduler allocates processor time to the task layer only when the partition is active. Hence the model of `PartitionScheduler` sends notification on the channels `enter_partition` and `exit_partition` to task schedulers when entering and leaving its partition, respectively.

The task layer contains a set of task models which are instantiated from two SWA templates `PeriodicTask` and `SporadicTask`. A task model describes an execution unit of airborne software. Since the tasks belonging to a partition are scheduled by its task scheduler, we define four channels `ready`, `release`, `sched` and `stop` as a set of scheduling commands to communicate between task templates and `TaskScheduler`. Moreover, the priority ceiling protocol is implemented by mutexes in task models to deal with intra-partition synchronization.

The communication layer comprises four TA templates: `UDPLayer`, `VLink`, `EndSystem` and `SwitchPort` in accordance with the structure of AFDX protocol stack. They jointly calculate the end-to-end delay of inter-partition communication through an AFDX network. When sending a message to an ARINC-653 port, the source task notifies the model of `UDPLayer` via a channel `pmsg`. `UDPLayer` transfers messages from the task layer to their corresponding VLs. In the link layer, two templates `VLink` and `EndSystem` model the latency of delivery through a transmitting ES. `VLink` realizes a traffic shaping function and shapes the flow of a VL to send no more than one frame in each interval of BAG. The `EndSystem` multiplexes the regulated flows coming from different VLs within a common end system. The channel `vl` synchronizes the execution of `UDPLayer` and `VLink`. Along the logical path of a VL, `SwitchPort` calculates the latency of queuing and forwarding frames at an output port of switches. The channel `frame` notifies the destination end system or switch port of the arrival of a frame. Additionally, two types of shared variables *Port* and *Buffer* model the counters of ARINC-653 ports and the queues at switches' ports, respectively.

In this framework, we verify the three following schedulability properties of DIMA systems:

- All the tasks meet their deadlines in each partition.
- The refresh period of any sampling port is guaranteed.
- The overflow from any queuing ports is avoided.

C. Model-based Analysis Method

On the basis of the ~~above~~ models described in the next section, we present the procedure for our schedulability analysis, which combines symbolic and statistical model checking. Fig. 7 shows the four steps in the procedure:

1) Scheduling configuration is encoded into the UPPAAL model as constant structure arrays.
2) We perform hypothesis testing of SMC for the model to falsify non-schedulable configuration rapidly.

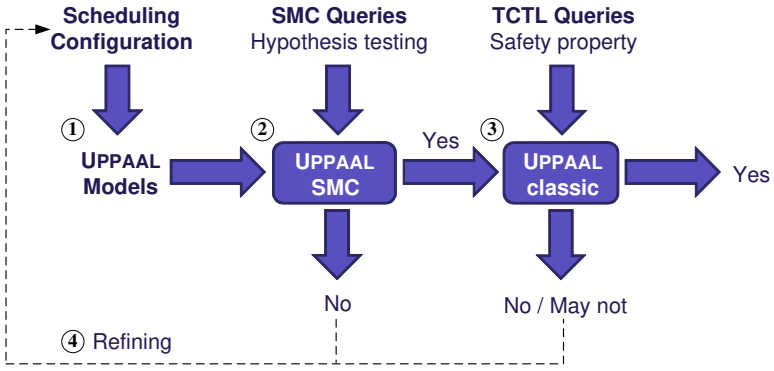


Fig. 7 Procedure for schedulability analysis

3) If the model goes through the SMC test, its schedulability should be verified by symbolic model checking.

4) We refine the configuration that fails in steps 2) or 3) and restart with the new configuration in step 1).

When we apply symbolic model checking to the analysis of a DIMA system, the schedulability constraints are expressed and verified as a safety property of SWA models. We add a set of *error locations* and a boolean variable `error` with the initial value `False` to UPPAAL templates for this purpose. Once the schedulability is violated, the related model will transfer to one of the error locations and assign the value `True` to `error` immediately. Thus, the schedulability is replaced with this safety property φ:

$$\texttt{A[] not error}, \tag{1}$$

which belongs to a simplified subset of TCTL (Timed Computation Tree Logic) [14] used in UPPAAL.

According to the size of state space, we choose either a global or compositional analysis. The system models with small size can be handled by the *global analysis* where all the constituent elements of a complete system are instantiated and checked directly. Nevertheless, most concrete systems have larger state space, thereby making the global analysis infeasible. To reduce the state space in this case, we perform a *compositional analysis* which check each partition including its environment individually. A set of message interface automata is built to model the environment for a partition.

The schedulability can be obtained from the satisfaction of φ, i.e. the result "Yes" from UPPAAL Classic in Fig. 7. However, since the symbolic model checking of UPPAAL for SWA introduces a slight over-approximation [10], we cannot conclude non-schedulability from the other results "No" or "May not" with certainty. Therefore, we derive non-schedulability from SMC testing rather than from the verification of φ.

Considering the scalability of SMC, we only use a global analysis in UPPAAL SMC. The schedulability of a complete

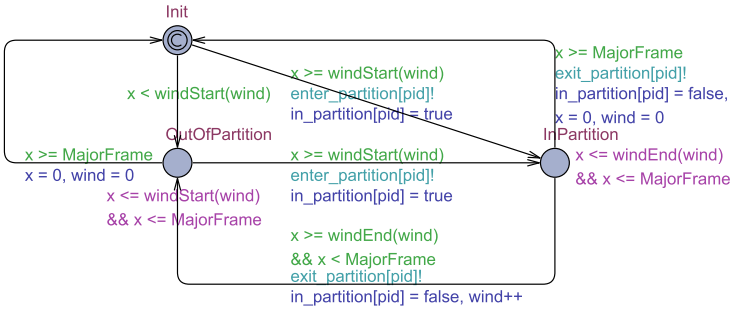


Fig. 8 PartitionScheduler model

avionics system is described as following queries of hypothesis testing:

$$\texttt{Pr[<= M](<> error) <=}\ \theta, \tag{2}$$

where M is the time bound on the simulations and θ is a very low probability. Since UPPAAL SMC approximates the answer using simulation-based tests, we can falsify non-schedulable configuration (i.e. the SMC result "No" in Fig. 7) rapidly by finding counter-examples but identify schedulable ones only with high probability $(1-\theta)$ (i.e. the SMC result "Yes" in Fig. 7). Hence, the configuration that goes through the SMC tests should be validated by symbolic model checking to ensure the schedulability of the corresponding system.

D. UPPAAL Models

In this section we present the UPPAAL templates used to perform the analysis. A zip file containing all the models can be downloaded from `http://people.cs.aau.dk/~ulrik/submissions/091437/models.zip`. The section is divided into three parts each representing a layer of the system as in Fig. 6. The parts then go into details of the UPPAAL templates used in that particular layer.

1. Scheduling Layer Models

PartitionScheduler template: In the scheduling layer, a partition is activated only during its partition windows within every major time frame. We build a TA model `PartitionScheduler`(See Fig. 8) to provide the description of temporal resources for a particular partition.

The template declarations in UPPAAL support the execution of a `PartitionScheduler` model. The parameter `pid` of `PartitionScheduler` is the identifier of its partition and the partition schedule is recorded in an array of structures `PartitionWindows`. Each element in the array contains two integer fields `offset` and `duration`, where `offset` is the start time of a partition window and `duration` denotes the duration of this window. By reading `PartitionWindows Table` from the declarations, the functions `winStart` and `winEnd` with the same integer parameter `wind` return the

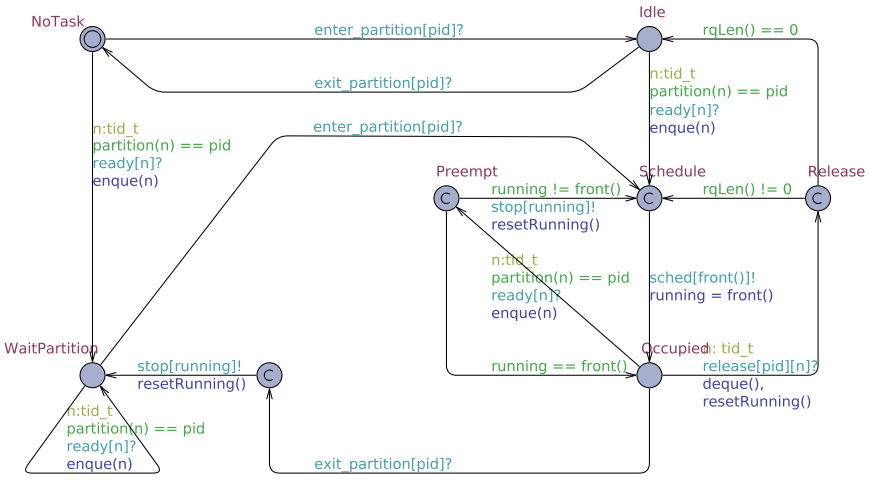


Fig. 9 AMPTaskScheduler model

start time and the end time of the *wind*th partition window, respectively. The integer constant `MajorFrame` stands for the major time frame, and the clock `x` measures time within every `MajorFrame`. In the template, all the guards and invariants use `x` to control the transitions between locations.

There are three locations in a `PartitionScheduler` model. The initial location `Init` represents a conditional control structure that determines the next location at the start of a major time frame. If a partition window and the major time frame start simultaneously, the model will move to the location `InPartition`. Otherwise, it will enter the location `OutOfPartition`. Within a major time frame, the model keeps traveling between `InPartition` and `OutOfPartition` according to whether or not the current time is in a partition window. For any time from the initial instant, if the `PartitionScheduler` model of `pid` enters a new partition window, it will move to the location `InPartition`, and notify the unique task scheduler model in `pid` through the output channel `enter_partition`. On the contrary, if the `PartitionScheduler` leaves its current partition window, it will move to the location `OutOfPartition`, and send notification to the task scheduler model through the output channel `exit_partition`.

AMPTaskScheduler template: For any partition in AMP configuration, there is a preemptive FP task scheduler that runs on a particular processor core while the partition is active. The behavior of the task scheduler is depicted in the TA template `AMPTaskScheduler` (See Fig. 9). Its partition is identified by the only template parameter `pid`.

The model of `AMPTaskScheduler` receives notification from the `PartitionScheduler` model through two channels `enter_partition` and `exit_partition`, and uses the channels `ready`, `release`, `sched` and `stop` as scheduling commands to manage the tasks in the partition `pid`. If there is a task becoming ready to run or relinquishing the processor, the task model will send its `AMPTaskScheduler` model a `ready` or `release` command, respectively. `AMPTaskScheduler` maintains a ready queue that keeps all the tasks ready and waiting to run, and always allocates the processor to the first task with the highest priority in the ready queue. If a new task having a higher priority than any tasks in the ready

Table 1 Major locations in task scheduler

Location	Partition windows		Ready tasks	
	Outside	Inside	0	> 0
NoTask	√		√	
Idle		√	√	
WaitPartition	√			√
Occupied		√		√

queue is ready, AMPTaskScheduler will insert the task into the ready queue, interrupt the currently running task via the channel stop and schedule the new selected task via the channel sched. The task identifier is delivered by the offset of channel arrays in the synchronization between AMPTaskScheduler and the task layer.

The ready queue is implemented by the integer array rq which contains a sorted set of task identifiers in priority order. The tasks with identical priority are served in order of readiness. The function rqLen returns the number of the tasks in rq. We use the function enque to insert an identifier of a new task into the ready queue rq and reorder the tasks in the queue. The function deque removes the first element from the ready queue. The first element in rq, i.e. the currently running task, is returned from the function front and recorded in the integer variable running.

According to whether the current time is in the partition windows as well as to the number of the tasks in the ready queue, we create four major locations listed in Table 1. These four locations cover all situations, where the model must be at one of these locations for any time from the initial instant. In contrast, all the other locations of the template are committed and utilized to realize conditional branches or atomic action sequences.

The template AMPTaskScheduler is an event-driven model. It always stays at one of the major locations and reacts to a particular set of input channels, each of which represents a type of external event. The event-handling functionality of ~~those~~ these four major locations is below:~~listed as follows.~~

- **NoTask** reacts to the set of input channels {enter_partition, ready}. At the location NoTask, the partition pid is not active and the ready queue rq is empty. Entering the partition pid will lead AMPTaskScheduler to the location Idle via the channel enter_partition. When a new task becomes ready, the scheduler will add the task to the ready queue and move to the location WaitPartition.
- **Idle** reacts to the set {exit_partition, ready}. Although the partition pid is active at the location Idle, there is no task being executed because of the empty ready queue rq. Leaving the partition leads to the input of exit_partition, which makes the model return to the location NoTask. However, once a task is ready to run at Idle, the model will insert the task into rq and then schedule the first task in rq via the output channel sched.
- **WaitPartition** reacts to the set {enter_partition, ready}. At the location WaitPartition, there is no task being executed despite the existence of tasks being ready in rq, because ~~for the time is out of~~ the partition pid is inactive.

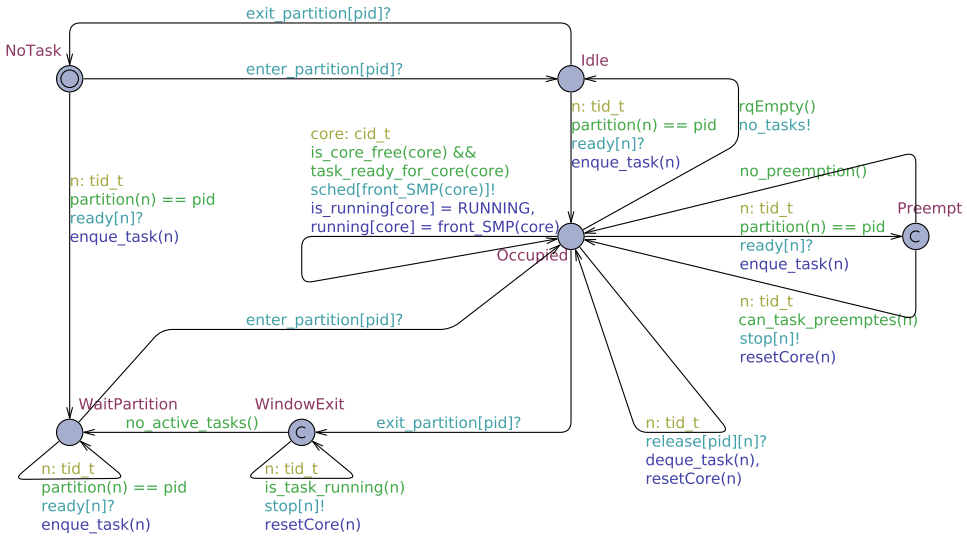


Fig. 10 SMPTaskScheduler model

Each of the tasks going to ready state must be recorded in `rq`. After entering the next partition window of `pid`, the model schedules the first task in `rq` to run.

- **Occupied** reacts to the set {`exit_partition`, `ready`, `release`}. At the location `Occupied`, the time is in a partition window of `pid` and there is at least one task in the ready queue `rq`. First, `AMPTaskScheduler` will react immediately to the input channel `exit_partition` and stop the execution of tasks when leaving the current partition window. Second, when a new task is ready, `AMPTaskScheduler` will handle the input action `ready` and thus add the task to `rq`. After that, if the running task recorded in `running` is not the first element in `rq` returned from `front`, the model will promptly emit the output `stop[running]!` and `sched[front()]!` to perform preemption. Third, when the running task relinquishes the processor, `AMPTaskScheduler` will receive notification from the input channel `release` and remove the task from `rq`. At ~~the~~ that moment, if the ready queue is not empty, the highest-priority task will be rescheduled via the channel `sched[front()]`.

SMPTaskScheduler template: For the SMP configuration, we need a second task scheduler that can manage a multi-core partition. Consequently, `SMPTaskScheduler` must be able to manage multiple running tasks running on different cores. The idea of this scheduler is very similar to the `AMPTaskScheduler`, but there are some elementary differences between the two templates. The `SMPTaskScheduler` is shown in Fig. 10, and its main difference is that the location `Occupied` contains a lot of behavior which was originally ~~divined~~ divided between several locations in `AMPTaskScheduler`. This location's outgoing edges each represent one of the following: (1) Schedule a task, (2) release a task, (3) ready a task, (4) entering `Idle` if no active tasks exits, and (5) exit partition.

Scheduling a task is done as soon as possible since `sched` is defined as an urgent channel. Therefore, a synchronization happens without delay if the guards evaluate to true. Part of the guard is to make sure that there is a task ready

for the core in question. The edge with release just removes the given task from the ready queue. For each core there is a corresponding ready queue, such that all active tasks are grouped by their affinity. The queues are arranged by decreasing priority such that the highest priority task is at the front. Whenever a task synchronize over ready, there is a possibility that a task is preempted. We know the affinity of the new ready task, and we therefore know on which core a preemption might occur. Since the tasks are grouped by affinity we just compare the running task with the front of the given core's ready queue. No preemption occurs if the front task is the same as the running task but is otherwise preempted in order for the core to be free for the new task.

If there are no active tasks in any ready queue the location Idle is entered by the use of no_tasks. This urgent channel is not used to synchronize with any other process but is a way to force the process to enter Idle when possible. The last edge from Occupied uses the channel exit_partition. As there might be several active tasks, we need to make sure that all tasks are preempted. In WindowExit all active tasks are preempted and only when that is completed the process enters WaitPartition. The main locations in SMPTaskScheduler are in the end the same as in AMPTaskScheduler which means that Table 1 applies for SMPTaskScheduler as well. Even the four major locations of SMPTaskScheduler react to the same sets of channels as in AMPTaskScheduler, and it is therefore following the same event-driven ~~structure~~ pattern.

2. Task Layer Models

We build two SWA templates PeriodicTask and SporadicTask in UPPAAL. Both templates share the same skeleton, so we take PeriodicTask as an example to sketch out the structure of a task model.

In the template, we define two normal clocks x and curTime and a stopwatch exeTime. The clock x measures the delays prescribed by the task type to calculate the release points of the task. The clock curTime is used to determine the start of the next task period and check if the deadline is missed. By contrast, the stopwatch exeTime measures the processing time during the execution of an abstract instruction that describes concrete task behavior, and will thus ~~progresses~~ only progress when the model is at the location Running.

Once the task is scheduled by TaskScheduler through the channel sched, it will start execution on the processor and move from the location Ready to ReadOp. For any task in the system, a sequential list of abstract instructions is implemented as the structure array op. By using an integer variable pc as a program counter, the task can fetch the next abstract instruction from op[pc] at the location ReadOp (See Fig. 11*).

According to the command in the abstract instruction currently read from op, the task model performs a conditional branch and moves from the location ReadOp to one of the locations that represent different operations. Therefore, the command set containing the following seven elements divides the rest of the template into seven corresponding parts.

- **COMPUTE Command:** If the model reads a COMPUTE command, it will (re)start the stopwatch exeTime and enter

*Due to the large size, the complete template is presented in Appendix C.

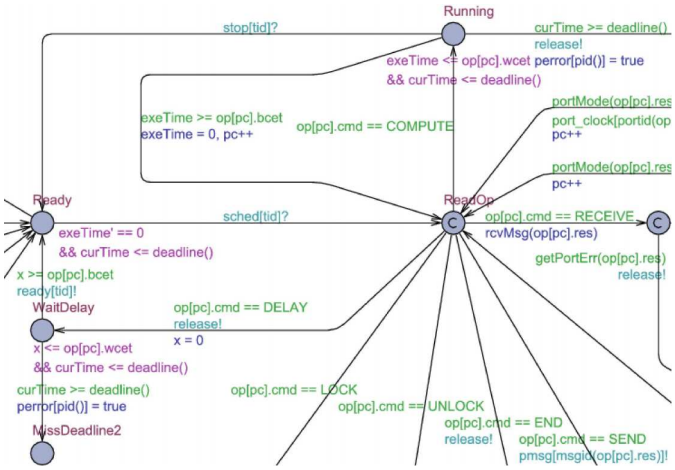


Fig. 11 Main structure of a task model

the location `Running`, which means that the processor is being occupied by the task and executing a computation instruction.

- **LOCK Command:** By reading a `LOCK` command, the task attempts to acquire the mutual exclusion lock that is specified by the `res` field of the instruction. The availability of a mutual exclusion lock depends on the priority ceiling protocol. If the lock is available, the task will acquire the lock and return to the location `ReadOp` immediately. Otherwise, the task will block itself and wait for the lock at the location `WaitResource`.
- **UNLOCK Command:** When fetching an `UNLOCK` command from `op`, the task releases the lock in the instruction and wakes up one of the tasks blocked on this lock. The woken task will leave the location `WaitResource` and enter the location `Ready` to wait for the next scheduling command.
- **DELAY Command:** The instruction with a `DELAY` command makes a task suspended at the location `WaitDelay` for a specified period of time. Thereafter, the task returns to `Ready`, waiting for its next scheduling command.
- **SEND Command:** The commands `SEND` and `RECEIVE` represent non-blocking message I/O operations among different partitions. When the task reads a `SEND` command with a resource identifer `op[pc].res`, the corresponding UDP/IP layer model will be notified of the message-sending operation through an output channel `pmsg[msgid(op[pc].res)]!`, where the offset `msgid(op[pc].res)` returns the identifier of the message type.
- **RECEIVE Command:** According to the transfer mode of the source port, there are two ways of processing a `RECEIVE` command. If the task receives a message from a queuing port, the model will call the function `rcvMsg` that decreases the counter of the source port. By contrast, the counter of a sampling port ~~does not be~~ isn't changed, but the task checks the source port's clock in the global clock array `port_clock` to ensure that the validity of received messages is consistent with the required refresh period of the source port. The clock of a sampling port is reset by the communication layer only when a new message arrives in the port.
- **END Command:** The command `END` denotes the accomplishment of the current job in this task period. The task

will relinquish the processor through the channel `release` and stay at the location `WaitNextRelease` until the next period starts.

3. Communication Layer Models

The communication layer consists of four TA templates: `UDPLayer` receives message sending requests from the task layer and transfers messages from ARINC-653 ports to their target VLs through the UDP/IP layer. `VLink` regulates transmitted flows to ensure a BAG (bandwidth allocation gap) interval between two consecutive frames. `EndSystem` using a First-In-First-Serve (FIFO) scheduler multiplexes the different flows coming from the `VLink`. `SwitchPort` acting as the queue inside a switch output port, forwards frames according to a forwarding table. We first detail the template `VLink` and then briefly describe the other three templates due to their similar skeleton.

VLink template We create one `VLink` instance per VL. Its unique template parameter `vlid` is the identifier of a VL. The `VLink` models read their configuration from the array `vlink`, which contains a source port `src`, an array `dst` of destination ports, a forwarding table `ft`, an identifier `es` of the VL's transmitting ES, an integer field `BAG` that stands for the bandwidth allocation gap [4], an integer field `TxDelay` denoting the frame delay [4], etc.

The total delay through a VL is divided into technological and configuration latency. Technological latency is independent of traffic load, whereas configuration latency depends on system configuration and traffic load [4].

The technological latency is bounded on the interval [`TechMin`, `TechMax`] where `TechMin` and `TechMax` are declared as two integer constants. A clock `x` measures the nondeterministic technological latency at a location `TechDelay`. The configuration latency through a transmitting ES is divided into three parts: (1) the floating delay in waiting for the interval of BAG, (2) the jitter within each BAG, and (3) a fixed frame delay. The first delay arises from the traffic regulation of `VLink`. A clock `t` measures the first delay since the last output to `EndSystem`. The second jitter is caused by the interference from other VLs in the same transmitting ES [4]. Hence the jitter will be calculated by the model of `EndSystem`. The frame delay is finally added to cover the time taken to deliver a frame to the physical link.

As is depicted in Fig. 12, `VLink` obtains notification of packet-receiving on the input channel `vl`. At the initial location `Idle`, `VLink` waits for the first packet to arrive at the source port. On receiving this first packet, the model fetches it from the port and goes through the technological latency at the location `TechDelay`. Subsequently, the model sends a new frame by entering the location `Sending` and resets the clock `t` to start the latency calculation for a new BAG. Meanwhile, the VL identifier is added to the FIFO queue of the corresponding end system. Leaving the location `Sending` means `VLink` has sent a regulated frame to `EndSystem`.

According to the number of packets in the source port, `VLink` may wait for the next BAG or the next incoming packet after finishing a sending operation. First, if the model still has at least one packet in the source port to transmit, it will start the next sending procedure by entering `TechDelay` again. Since there is at least a BAG interval between two consecutive frames, `VLink` should wait for the start of the next BAG at location `Regulation` before entering `Sending`.

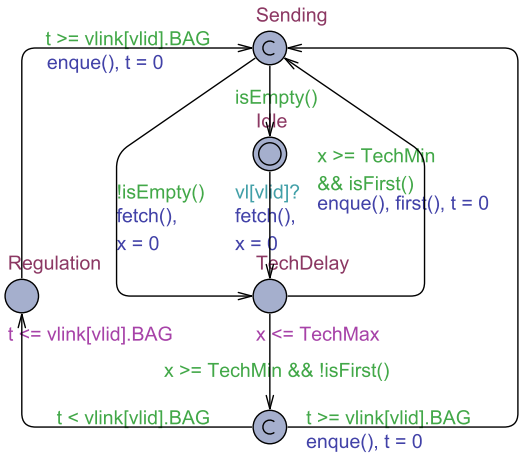


Fig. 12 VLink model

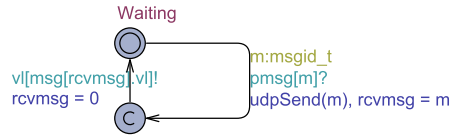


Fig. 13 UDP layer model

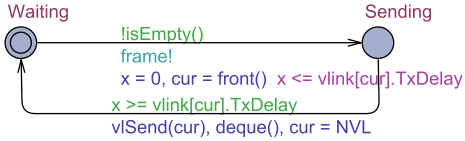


Fig. 14 End system model

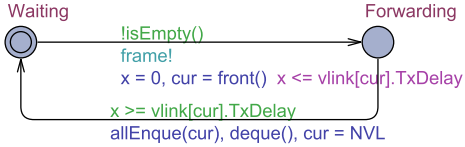


Fig. 15 Switch port model

Second, if the source port is empty, the model will stay at `Idle` until the next incoming packet arrives.

The other templates As shown in Fig. 13-15, the other three templates share ~~the~~ a similar structure. All of them have a `Waiting` location at which they wait for the next incoming packet from their upper layer. Once they are notified via an input channel or an increment of shared variables, they will enter another "Transmission" location and transfer the packet from the upper to the lower. The model returns to `Waiting` and repeats this process after finishing the transit operation. In addtion, `EndSystem` and `SwitchPort` use an urgent channel `frame` to check their own FIFO buffers in time. They choose destination buffer(s) on the basis of the forwarding table `ft` of VLs. The function `enque` adds the VL identifer of a frame to the next buffer, while `deque` removes the front from ~~the~~ its own buffer.

V. Compositional Analysis

This section focuses on the compositional schedulability analysis, which verifies local properties of individual partitions by symbolic model checking, infers that they still hold in the complete system, and finally deduces global properties of the system.

We adopt the paradigm of assume-guarantee reasoning [43] to implement the compositional analysis. The basic element of this paradigm is normally expressed as a triple $\langle\phi\rangle M\langle\varphi\rangle$, where ϕ and φ are logic formulas and M is a model. The triple is true if whenever M is a constituent of a system satisfying ϕ, the system is guaranteed to satisfy φ. Consider the system consisting of two components M and M'. A typical assume-guarantee rule is defined as

$$\frac{\langle true\rangle M'\langle\phi\rangle \quad \langle\phi\rangle M\langle\varphi\rangle}{\langle true\rangle M\|M'\langle\varphi\rangle} \tag{3}$$

where the environment M' of the component M guarantees the assumption ϕ of M. It concludes that the complete system satisfies φ.

A classic way [52] to realize such a paradigm is to provide a preorder $\preceq$ on the finite-state models that captures the notion of "more behaviors" and to use a logic whose semantics is consistent with the preorder. The preorder should not only preserve satisfaction of the logic formulas but also hold in composition operations of models. For example, if a formula is true for a model, it will also be true for any model that is smaller in the preorder. Additionally, satisfaction of a formula corresponds to being smaller than a tableau model of the formula in the preorder. Hence assumptions can be defined either as logic formulas or directly as finite-state models. Let A be the tableau of the assumption ϕ. The above assume-guarantee rule can be expressed as

$$\frac{M' \preceq A \quad M \| A \models \varphi}{M \| M' \models \varphi} \tag{4}$$

Considering the conciseness and convenience, we describe assumptions of each component as SWA models directly and implement assume-guarantee reasoning as the rule of Eq. (4).

In this section, we first define a preorder, timed selection simulation relation on SWA models. On the basis of its properties, we present the procedure for compositional analysis in the paradigm of assume-guarantee reasoning. A simplified avionics system exemplifies the use of our compositional analysis.

A. Timed Selection Simulation

We propose a notion of timed selection simulation relation to support assume-guarantee reasoning. Compared with some other abstraction relations like timed simulation [53] and timed ready simulation [54], timed selection simulation only abstracts a selected subset of actions from the concrete model. Applying timed selection simulation to the abstraction of a concrete system, one can pay attention to part of the system, individually model the behavior of each component, and thereby obtain a composite abstract model rather than a monolithic one.

Considering the semantic object $\mathcal{T}^A$ of an automaton $A \in \Omega$, we denote the *error states* of $\mathcal{T}^A$ by the set $\mathcal{E} = \{\langle l, u\rangle | l \in Err\}$ where Err is the error-location set of A. Thus, for any TIOTS $\mathcal{T} = \langle S, s_0, \Sigma, \rightarrow\rangle$, its error states are defined as a set $\mathcal{E} \subseteq S$, and the following function $g : S \rightarrow \{true, false\}$ indicates whether a state $s \in S$ has violated schedulability properties:

$$g(s) = \begin{cases} true & \text{if } s \in \mathcal{E} \\ false & \text{if } s \notin \mathcal{E}. \end{cases} \tag{5}$$

Given two compatible TIOTSs $\mathcal{T}_i, i \in \{1, 2\}$ with the error-state set $\mathcal{E}_i$, their composition $\mathcal{T}_1 \| \mathcal{T}_2$ has the error-state set $\mathcal{E}_{\mathcal{T}_1 \| \mathcal{T}_2} = \{\langle s_1, s_2\rangle | s_1 \in \mathcal{E}_1 \vee s_2 \in \mathcal{E}_2\}$ and the function $g(\langle s_1, s_2\rangle) = g(s_1) \vee g(s_2)$.

Based on the function $g(s)$, the formal definition of timed selection simulation is given as follows.

Definition 5 (Timed Selection Simulation) *Let* $\mathcal{T}_1 = \langle S_1, s_{1,0}, \Sigma_1, \rightarrow_1 \rangle$ *and* $\mathcal{T}_2 = \langle S_2, s_{2,0}, \Sigma_2, \rightarrow_2 \rangle$ *be two timed I/O transition systems with* $\Sigma_2 \subseteq \Sigma_1$. *Let R be a relation from* S_1 *to* S_2. *We call R a timed selection simulation from* $\mathcal{T}_1$ *to* $\mathcal{T}_2$, *written* $\mathcal{T}_1 \preceq \mathcal{T}_2$ *via R, provided* $(s_{1,0}, s_{2,0}) \in R$ *and for all* $(s_1, s_2) \in R$, $g(s_1) = g(s_2)$ *and*

1) *if* $s_1 \xrightarrow{a?} s_1'$ *for some* $s_1' \in S_1$, $a \in \Sigma_2$, *then* $\exists s_2' \in S_2$ *such that* $s_2 \overset{a?}{\Longrightarrow} s_2'$ *and* $(s_1', s_2') \in R$
2) *if* $s_1 \xrightarrow{a!} s_1'$ *for some* $s_1' \in S_1$, $a \in \Sigma_2$, *then* $\exists s_2' \in S_2$ *such that* $s_2 \overset{a!}{\Longrightarrow} s_2'$ *and* $(s_1', s_2') \in R$
3) *if* $s_1 \xrightarrow{a} s_1'$ *for some* $s_1' \in S_1$, $a \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau\}$, *then* $\exists s_2' \in S_2$ *such that* $s_2 \overset{\mathbf{0}}{\Longrightarrow} s_2'$ *and* $(s_1', s_2') \in R$
4) *if* $s_1 \xrightarrow{\epsilon(d)} s_1'$ *for some* $s_1' \in S_1$, $d > 0$, *then* $\exists s_2' \in S_2$ *such that* $s_2 \overset{\epsilon(d)}{\Longrightarrow} s_2'$ *and* $(s_1', s_2') \in R$.

Definition 6 (Timed Selection Simulation between SWA) *Let* $A_i, i \in \{1, 2\}$ *be stopwatch automata. We say that* $A_1 \preceq A_2$, *if and only if their corresponding timed I/O transition systems* $\mathcal{T}_i$ *satisfy* $\mathcal{T}_1 \preceq \mathcal{T}_2$.

We now give some necessary properties of timed selection simulation.

Theorem 1 (Preorder) *Timed selection simulation* $\preceq$ *is a preorder.*

For any automaton $A \in \Omega$, by construction, the reachability of its error locations is equivalent to that of the error states in the corresponding TIOTS $\mathcal{T}^A$. Hence the following theorem shows that timed selection simulation can preserve the satisfaction of the safety properties in the form of Eq.(1).

Theorem 2 (Property preservation) *Let* $\mathcal{T}_i, i \in \{1, 2\}$ *be timed I/O transition systems and* $\mathcal{E}_i$ *be the set of error states of* $\mathcal{T}_i$. *Given a safety property* $\varphi : \forall \neg reach(\mathcal{E}_i)$ *that any error states are not reachable, if* $\mathcal{T}_1 \preceq \mathcal{T}_2$ *and* $\mathcal{T}_2 \models \varphi$, *then* $\mathcal{T}_1 \models \varphi$.

Theorem 3 (Abstraction compositionality) *Let* $\mathcal{T}_i, i \in \{1, 2, 3\}$ *be timed I/O transition systems. If* $\mathcal{T}_1 \preceq \mathcal{T}_2$, $\mathcal{T}_1 \preceq \mathcal{T}_3$, *and* $\mathcal{T}_2$ *and* $\mathcal{T}_3$ *are compatible, then* $\mathcal{T}_1 \preceq \mathcal{T}_2 \| \mathcal{T}_3$.

Theorem 4 (Compositionality) *Let* $\mathcal{T}_i = \langle S_i, s_{i,0}, \Sigma_i, \rightarrow_i \rangle$, $i \in \{1, 2, 3, 4\}$ *be timed I/O transition systems. Suppose* $\mathcal{T}_1 \| \mathcal{T}_3$ *and* $\mathcal{T}_2 \| \mathcal{T}_4$ *are the parallel compositions of compatible timed I/O transition systems. If* (1) $\mathcal{T}_1 \preceq \mathcal{T}_2, \mathcal{T}_3 \preceq \mathcal{T}_4$, *and* (2) $O_1 \cap I_4 \subseteq \Sigma_2, I_2 \cap O_3 \subseteq \Sigma_4$, *then* $\mathcal{T}_1 \| \mathcal{T}_3 \preceq \mathcal{T}_2 \| \mathcal{T}_4$.

B. Procedure for Compositional Analysis

Our compositional analysis exemplifies the paradigm of assume-guarantee reasoning. Compared to the existing assume-guarantee approaches, our approach is based on the timed selection simulation relation that has the novel feature of abstraction compositionality. This property helps engineers generate the abstract model of a component automatically by combining a set of simple message interface automata. When a partition P_i is checked independently, these message interface automata describe the external behavior of the other partitions, serving as the assumptions of P_i in assume-guarantee reasoning.

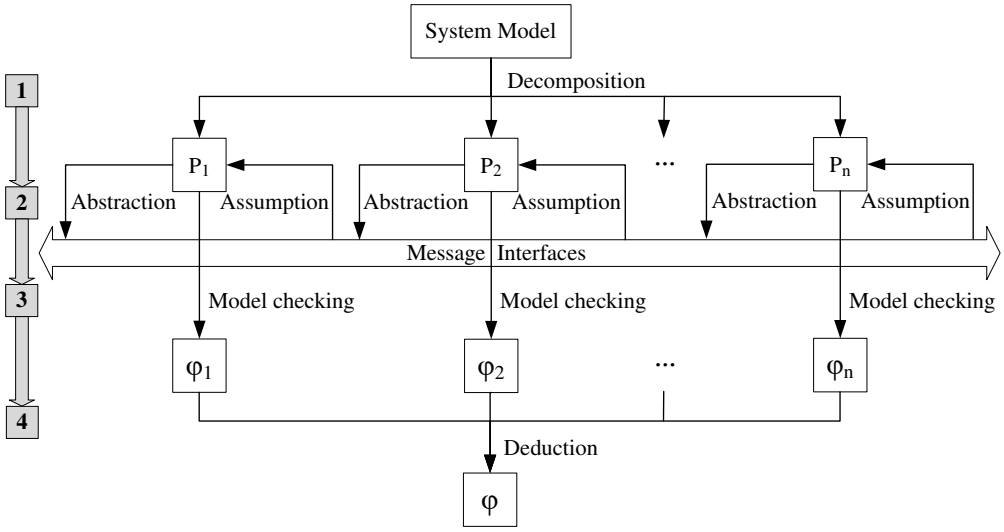


Fig. 16 Compositional analysis procedure

We apply the assume-guarantee rules like Eq. (4) to our compositional analysis, and describe the schedulability goal as the safety property φ of Eq. (1). As shown in Fig. 16, our compositional analysis is comprised of the following four steps:

1) *Decomposition:* The system is first decomposed into a set of communicating partitions modeled by TA and SWA. The global property φ is also divided into several local properties, each of which belongs to one partition.
2) *Construction of message interfaces:* We define message interfaces as the assumption and abstraction of the communication environment for each partition. In general, the templates of message interfaces should be built manually by the engineers.
3) *Model checking:* The local properties under the assumptions and the abstraction relations are verified by model checking.
4) *Deduction:* From the assume-guarantee rules, we finally derive the global property by combining all the local results.

The procedure can be performed automatically except for the first construction of message interfaces. We assume that a task never blocks while communicating with other partitions, which is commonly used in avionics systems [11, 22]. Otherwise a loop of communication dependency will cause circular reasoning, because the assumptions of a partition might be based on its own state recursively.

1. Decomposition

We first instantiate the templates of the UPPAAL modeling framework to construct the SWA model of a complete avionics system Λ. Given a template name `Template`, we define the set of its SWA instances in the system as {`Template`} and the composition of all the elements in {`Template`} as [`Template`]. Let $\Phi(A)$ be the set of SWA instances for any

model A. $\Phi(\Lambda)$ contains nine disjoint sets, each of which corresponds to one template in the modeling framework:

$$\begin{aligned}\Phi(\Lambda) =&\big\{\{\texttt{PartitionScheduler}\} \cup \{\texttt{AMPTaskScheduler}\} \cup \{\texttt{SMPTaskScheduler}\} \cup \{\texttt{PeriodicTask}\} \\ &\cup \{\texttt{SporadicTask}\} \cup \{\texttt{UDPLayer}\} \cup \{\texttt{VLink}\} \cup \{\texttt{EndSystem}\} \cup \{\texttt{SwitchPort}\}\big\}.\end{aligned} \tag{6}$$

Hence the system Λ is described as a composite SWA model

$$\begin{aligned}\Lambda =&[\texttt{PartitionScheduler}]|||[\texttt{AMPTaskScheduler}]|||[\texttt{SMPTaskScheduler}]|||[\texttt{PeriodicTask}] \\ &|||[\texttt{SporadicTask}]|||[\texttt{UDPLayer}]|||[\texttt{VLink}]|||[\texttt{EndSystem}]|||[\texttt{SwitchPort}].\end{aligned} \tag{7}$$

Assume that there are n constituent partitions in a system. Let $P_i, i \in \{1, 2, \ldots, n\}$ be the SWA composite model of the ith partition. We also refer to this partition as P_i if its meaning is not ambiguous. P_i consists of all the SWA instances of the scheduling and task layer within the partition:

$$P_i = [\texttt{PartitionScheduler}]_i|||[\texttt{AMPTaskScheduler}]_i|||[\texttt{SMPTaskScheduler}]_i|||[\texttt{PeriodicTask}]_i|||[\texttt{SporadicTask}]_i \tag{8}$$

where the constituent models belong exclusively to P_i. Thus the scheduling and task layer of the system are divided into n disjoint composite model of partitions $P_i, i \in \{1, 2, \ldots, n\}$, and $\bigcap_{i\in\{1,2,\ldots,n\}} \Phi(P_i) = \emptyset$.

By contrast, all the partitions share the same model F of the AFDX network facilities in the communication layer:

$$F = [\texttt{UDPLayer}]|||[\texttt{VLink}]|||[\texttt{EndSystem}]|||[\texttt{SwitchPort}] \tag{9}$$

which cannot be decomposed like the above layers. However, when only considering the communication environment of one partition P_i, we traverse the forwarding table `ft` of VLs and recursively extract models of the network facilities that affect the message transmission to P_i directly or indirectly. In doing so, F is decomposed into n intersecting composite model $F_i, i \in \{1, 2, \ldots, n\}$:

$$F_i = [\texttt{UDPLayer}]_i|||[\texttt{VLink}]_i|||[\texttt{EndSystem}]_i|||[\texttt{SwitchPort}]_i \tag{10}$$

where it is possible that $\bigcap_{i\in\{1,2,\ldots,n\}} \Phi(F_i) \neq \emptyset$. We define the composite model of the partition P_i with its network facilities F_i as $P_i^* = P_i \| F_i$.

Let Err_i be the error-location set of P_i. The safety property φ_i: `A[]` $\neg(\bigvee_{loc\in Err_i} loc)$ denotes the schedulability of P_i. The global property φ is therefore written as $\varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_n$, and the goal of our schedulability analysis is

expressed as the verification problem:

$$\Lambda \models \varphi. \tag{11}$$

Since the error-location set Err_i is only allowed to be manipulated by P_i, the problem can be further divided into n satisfaction relations:

$$P_i^* \parallel (\|_{j=1,j\neq i}^{n} P_j) \models \varphi_i, \;\; i \in \{1, 2, \ldots, n\}. \tag{12}$$

When handling the property φ_i, we also write P_i instead of P_i^* without ambiguity. These n subproblems are thus written as

$$P_1 \| P_2 \| \cdots \| P_n \models \varphi_i, \; i \in \{1, 2, \ldots, n\}. \tag{13}$$

In an ideal compositional way, we should check each partition model P_i independently for the corresponding *local property* φ_i rather than the original verification problem with a larger system model. However, the communication environment of P_i, which denotes the behavior that P_i receives messages from other partitions, may affect the satisfaction of the schedulability property φ_i. Hence when performing the verification for partition P_i, one needs to give the *assumptions* of its communication environment and verifies the local property φ_i under these assumptions.

2. Construction of message interfaces

A set of TA models is created to describe the message-sending behavior of a partition. Each of the TA is called a *message interface* of this partition and associated with a particular message type. Suppose there are a number of messages sent from partition P_j to another partition P_i and their corresponding message interfaces make up a composite TA model $A_{i,j}$. When we analyze P_i in the compositional way, it should be safe for $A_{i,j}$ to replace P_j. Hence, we say that a message interface of P_j is an *abstraction* of P_j.

Our abstraction of the message delivery between a partition and its underlying network is modeled using the synchronization between SWA models. An action of the synchronization represents a specific message types. Let $\Sigma_i = I_i \oplus O_i$ be the action set of a composite model for any partition P_i. An action $a_k \in I_i$(resp. $a_k \in O_i$) denotes that P_i receives(resp. sends) messages with the type msg_k from(resp. to) other partition(s). The symbol $j \triangleright i$ represents the condition that there exists a partition P_j sending messages to P_i via an action set $O_{j\rightarrow i} \subseteq I_i \cap O_j$.

Definition 7 (Message Interface) *Let O_i be the output action set of a stopwatch automaton $P_i \in \Omega$. For any output action $a_k \in O_i$, the timed automaton A_i^k with an action set $\Sigma_i^k = O_i^k = \{a_k\}$ is a message interface of P_i if and only if there exists a timed selection simulation relation $\preceq$ on Ω such that*

$$P_i \preceq A_i^k. \tag{14}$$

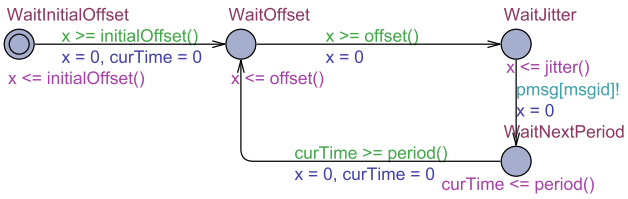


Fig. 17 Template of periodic message interface

We build message interfaces on the basis of the message patterns described in section III.C. Fig. 17 shows a template of the message interface that sends messages periodically via the action array `pmsg`. Then we make an automatized binary search for the interface's parameters such as `jitter` in the template and meanwhile check the satisfaction of timed selection simulation relation.

The message interfaces can serve as the assumptions of the communication environment of a partition. The composition $A_{i,j}$ of the message interfaces A_j^k for all $a_k \in O_{j\rightarrow i}$ provides P_i with a "complete" abstraction of P_j, which models the behavior of all the output actions from P_j to P_i. According to the abstraction compositionality (Theorem 3) of the preorder $\preceq$, we have

$$P_j \preceq A_{i,j}. \tag{15}$$

Considering all the partitions except P_i in the system, we describe the communication environment of P_i as the composite model $\|_{j=1,j\neq i}^{n} A_{i,j}$.

3. Model checking

In the third step, the local property φ_i of P_i under assumption $\|_{j=1,j\neq i}^{n} A_{i,j}$ can be verified by model checking. We denote these n subproblems by

$$P_i \parallel (\|_{j=1,j\neq i}^{n} A_{i,j}) \models \varphi_i \quad i \in \{1, 2, \ldots, n\}. \tag{16}$$

Normally, $A_{i,j}$ in Eq.(16) has a much smaller model size than its corresponding partition model P_j in Eq.(13). Thus, the compositional approach allows us to verify a simpler abstract partition model instead of a complex concrete system model including the details about all the partitions.

In addition, we capture the computation time of each task as an interval between a best-case and worst-case execution time. When analyzing the schedulability of a partition, the model-checker explores all scheduling decisions that can be made in such an interval, and hence also examines possible cases of scheduling timing anomalies [55].

4. Deduction

We derive the global property φ by combining n local results in the last step. For any schedulable system, each property φ_i should be concluded from the satisfaction of Eq.(16) under assumptions and all the abstraction relations

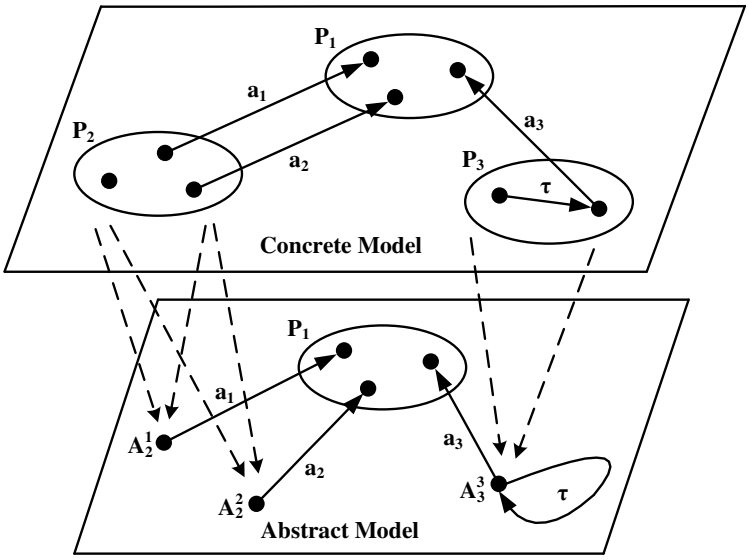


Fig. 18 Abstraction relations in the example

of Eq.(15). According to the compositionality (Theorem 4) and property preservation (Theorem 2) of timed selection simulation, we have the following assume-guarantee rule:

$$\frac{\bigwedge_{\{j\,|\,j\triangleright i\}} P_j \preceq A_{i,j} \qquad P_i \parallel (\|_{j=1,j\neq i}^{n} A_{i,j}) \models \varphi_i}{P_1\|P_2\|\cdots\|P_n \models \varphi_i} \tag{17}$$

~~Note that this assume-guarantee rule only provides a sufficient schedulability condition, for abstract message interfaces might slightly over-approximate the external behavior of a partition.~~

C. An Example of Assume-guarantee Reasoning

A simplified DIMA system exemplifies the reasoning procedure. In this example, the system consists of three partitions $P_i, i \in \{1, 2, 3\}$, each of which is able to communicate with other partitions directly via a set of messages. We write the global property φ as the conjunction of three local properties φ_i of P_i. Accordingly, the goal of the verification problem is to check

$$P_1\|P_2\|P_3 \models \varphi_1 \wedge \varphi_2 \wedge \varphi_3. \tag{18}$$

From Eq.(13), this problem can be replaced with three subproblems:

$$P_1\|P_2\|P_3 \models \varphi_i, i \in \{1, 2, 3\}. \tag{19}$$

Without loss of generality, we take the verification of φ_1 for example to show how the model-checking and deduction

are carried out in the following steps.

Assume that P_2 sends P_1 two types of messages, msg_1 and msg_2, via two actions a_1 and a_2 respectively, and P_3 sends P_1 only a msg_3 with action a_3. We create one message interface $A_j^k, j \in \{2,3\}$(like Eq.(14)) for each message type $msg_k (k \in \{1,2,3\})$ received by P_1 in the system. The abstraction relations from Eq.(14) can be expressed as

$$P_2 \preceq A_2^1,\ P_2 \preceq A_2^2,\ P_3 \preceq A_3^3. \tag{20}$$

The dashed-line arrows in Fig. 18 depict these abstraction relations. From abstraction compositionality of the preorder $\preceq$, we can obtain

$$P_2 \preceq A_2^1 \| A_2^2,\ P_3 \preceq A_3^3. \tag{21}$$

Then, from reflexivity and compositionality of the preorder $\preceq$, the composite model of the system satisfies

$$P_1 \| P_2 \| P_3 \preceq P_1 \| A_2^1 \| A_2^2 \| A_3^3. \tag{22}$$

Note that when we apply the compositionality to checking a partition P_i, any output actions sent to P_i will never be removed in abstraction relations (Eq.(21)), which satisfies the condition (2) of theorem 4.

With Eq.(22), we have from property preservation of the abstraction relation $\preceq$ that *if*

$$P_1 \| A_2^1 \| A_2^2 \| A_3^3 \models \varphi_1,\ \textit{then} \tag{23}$$

$$P_1 \| P_2 \| P_3 \models \varphi_1. \tag{24}$$

Since Eq.(24) covering all three partitions in the system has a higher complexity than Eq.(23), the techniques of model checking can be adopted to verify the simpler problem Eq.(23) instead of the original goal Eq.(24). The same steps will be repeated for local properties φ_2 and φ_3.

Consequently, we conclude all the local results of (19) according to the reasoning process from Eq.(20) to Eq.(24). When we analyze the partition P_1 and its communication environment, the local result of Eq.(24) can be deduced from Eq.(20) and Eq.(23) in the following assume-guarantee rule.

$$\frac{\begin{array}{r} P_2 \preceq A_2^1 \wedge P_2 \preceq A_2^2 \wedge P_3 \preceq A_3^3 \\ P_1 \| A_2^1 \| A_2^2 \| A_3^3 \models \varphi_1 \end{array}}{P_1 \| P_2 \| P_3 \models \varphi_1} \tag{25}$$

The local results are then combined to constitute the global result of Eq.(18).

Note that this assume-guarantee rule only provides a sufficient schedulability condition. Actually, a false negative result might be given conservatively even if the system configuration is schedulable. The conservativeness of our compositional approach stems from two conditions of over-approximation:

- The first condition of false negatives is due to the slight over-approximation used in the SWA verification algorithm inside the UPPAAL verification engine[10]. This conservativeness is inevitable but can be mitigated by the step of SMC falsification described in section IV.C. Therefore, a larger time bound M or a lower probability θ of the Eq. 2 is likely to be adopted in this case.
- The second over-approximation is caused by the abstraction of message interface automata, for they might slightly over-approximate the external behavior of partitions. In this paper, the message interfaces are designed as the automaton shown in fig. 17 but not confined to it, which sends one message within a fixed period like a task. If the assumption of message-sending behavior of tasks is not precise, the final message interface automata will occupy more time to send a possibly submitted message than the task actually needs. In this case, the compositional method might also return a false negative result. The degree of this conservativeness depends on engineers' experience in practice.

VI. Case Study

This section demonstrates the schedulability analysis of an avionics system which combines the workload in [11] and the AFDX configuration in [34].

As shown in Table 2, the workload is comprised of 5 partitions ($P_1 - P_5$), which then contains a total of 18 periodic tasks and 4 sporadic tasks. The type of a task depends on its *release* interval. A periodic task has a deterministic period, whereas the release time of a sporadic task is bounded by a minimum separation. The execution of a task is characterized as a sequence of *chunks*. Each chunk has a lower and upper bound on *execution time* (modeled as a non-deterministic choice), a set of potentially required resources and message-passing operations. There are 3 intra-partition locks, as shown in column *mutex*, and 4 inter-partition message types in the task set. The columns *output* and *input* indicate transfer direction of messages. According to the operations and resources required by chunks, we convert each chunk into a subsequence of the abstraction instruction sequence (*Receive*, *Lock*, *Compute*, *Unlock*, *Send*, *End*) in the UPPAAL task model.

Considering the inter-partition messages in the workload, we assign each message type (Msg_i) with a correnspoding unique VL (V_i). The messages of Msg_1 and Msg_2 are handled at the refresh period $50ms$ in sampling ports. Msg_3 and Msg_4 are configured to operate in queuing ports, each of which can accommodate a maximum of one message. The AFDX configuration in Table 3 is based on the case of [34]. The column *Length* indicates the length of a message sent from an ARINC-653 partition. For any VL in the configuration, the columns *BAG* and L_{max} denote its Bandwidth Allocation Gap and Maximum packet Length, respectively. The source and destination partition(s) are given in the

Table 2 Workload of the avionics system [11, 22](Times in milliseconds)

No.	Task	Release	Offset	Jitter	Deadline	Priority	Execution Chunks			
							Time	Mutex	Output	Input
P_1	Tsk_1^1	[25,25]	2	0	25	2	[0.8,1.3]	-	-	-
							[0.1,0.2]	-	-	-
	Tsk_2^1	[50,50]	3	0	50	3	[0.2,0.4]	-	Msg_1	-
	Tsk_3^1	[50,50]	3	0	50	4	[2.7,4.2]	-	-	-
	Tsk_4^1	[50,50]	0	0	50	5	[0.1,0.2]	Mux_1^1	-	-
	Tsk_5^1	[120,∞)	0	0	120	6	[0.6,0.9]	-	-	-
							[0.1,0.2]	Mux_1^1	-	-
P_2	Tsk_1^2	[50,50]	0	0.5	50	2	[1.9,3.0]	-	-	-
	Tsk_2^2	[50,50]	2	0	50	3	[0.7,1.1]	-	Msg_2	-
	Tsk_3^2	[100,100]	0	0	100	4	[0.1,0.2]	Mux_1^2	-	-
	Tsk_4^2	[100,∞)	10	0	100	5	[0.8,1.3]	-	-	-
							[0.2,0.3]	Mux_1^2	-	-
P_3	Tsk_1^3	[25,25]	0	0.5	25	2	[0.5,0.8]	-	-	Msg_1
	Tsk_2^3	[50,50]	0	0	50	3	[0.7,1.1]	-	-	Msg_2
	Tsk_3^3	[50,50]	0	0	50	4	[1.0,1.6]	-	-	Msg_3
	Tsk_4^3	[100,∞)	11	0	100	5	[0.7,1.0]	-	-	-
							[0.1,0.3]	-	-	-
P_4	Tsk_1^4	[25,25]	3	0.2	25	2	[0.7,1.2]	-	-	-
	Tsk_2^4	[50,50]	5	0	50	3	[1.2,1.9]	-	Msg_3	Msg_1
	Tsk_3^4	[50,50]	25	0	50	4	[0.1,0.2]	-	-	Msg_4
	Tsk_4^4	[100,100]	11	0	100	5	[0.7,1.1]	-	-	-
	Tsk_5^4	[200,200]	13	0	200	6	[3.7,5.8]	-	-	-
P_5	Tsk_1^5	[50,50]	0	0.3	50	1	[0.7,1.1]	-	-	Msg_1
	Tsk_2^5	[50,50]	2	0	50	2	[1.2,1.9]	-	Msg_4	Msg_2
	Tsk_3^5	[200,200]	0	0	200	3	[0.4,0.6]	-	-	-
							[0.2,0.3]	Mux_1^5	-	-
	Tsk_4^5	[200,∞)	14	0	200	4	[1.4,2.2]	-	-	-
							[0.1,0.2]	Mux_1^5	-	-

columns *Source* and *Destination*, respectively.

Fig. 19 illustrates the distributed deployment of the workload. We consider 3 ARINC-653 modules connected by an AFDX network, and allocate each partition to one of the modules. The module M_1 accommodates P_1 and P_2, the

Table 3 AFDX configuration in the case study (Times in milliseconds and sizes in bytes)

Message	Length	VL	BAG	L_{max}	Source	Destinations
Msg_1	153	V_1	8	200	P_1	P_3,P_4,P_5
Msg_2	953	V_2	16	1000	P_2	P_3,P_5
Msg_3	453	V_3	32	500	P_4	P_3
Msg_4	153	V_4	32	200	P_5	P_4

module M_2 executes P_3 and P_5, and the partition P_4 is allocated to M_3. There are 4 VLs V_1-V_4 connecting 3 ESs across 2 switches S_1 and S_2 in the AFDX network. The arrows above VLs' names indicate the direction of message flow.

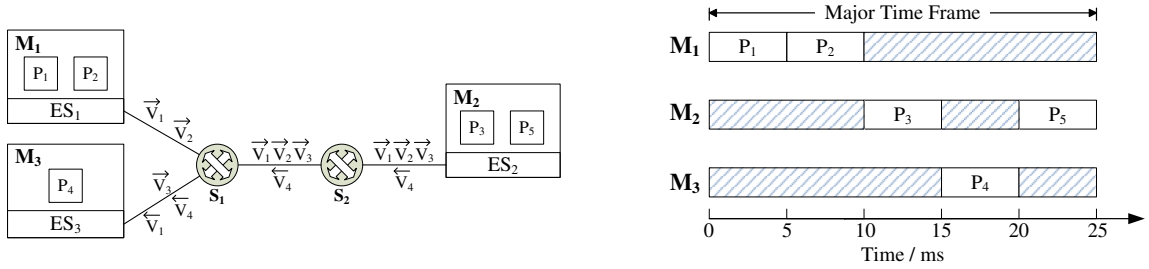


Fig. 19 Distributed avionics deployment and partition schedules (Times in milliseconds)

A. Experiment 1 in AMP configuration

We first consider an AMP configuration that equips each of its processor cores with one partition. Although the sample avionics workload[11, 22] is designed for single-core processor platforms, the AMP multi-core deployment does not change the sequential execution of these legacy applications. We update the module M_1 by installing a dual-core processor instead, while single-core processors remain in the modules M_2 and M_3. Fig. 19 gives the partition schedules, which fix a common major time frame MF at $25ms$ and allocate $5ms$ to each partition within every MF. All the partition schedules are enabled at the same initial instant and their clocks are always synchronized. The scheduling configuration keeps the temporal order of the partitions in [11]. Hence the partition schedules contain five disjoint windows $\langle P_1, 0, 5\rangle$, $\langle P_2, 5, 5\rangle$, $\langle P_3, 10, 5\rangle$, $\langle P_4, 15, 5\rangle$, and $\langle P_5, 20, 5\rangle$, where the second parameter is the offset from the start of MF and last the duration.

After combining all the models of the system, we executed the schedulability analysis in Uppaal. We set the timebound as $M = 1.0 \times 10^5$ microseconds and the probability threshold as $\theta = 0.001$ for Eq.(2). The experiment was performed on the Uppaal 4.1.19 64-bit version and an Intel Core i5-4590 processor.

Results of the Analysis

The result (The AMP case in Table 4) shows that the above scheduling configuration fails the SMC test and thus is non-schedulable. The following classical MC is not required but listed for comparison. We can explore the cause of

Table 4 Experiment results (Result), execution time (Time/seconds) and memory usage (Memory/MB)

Case	Method	Range	Result	Time	Memory
AMP	SMC	System	No	5.99	50
	MC	P_1	Yes	7.26	171
		P_2	Yes	0.92	52
		P_3	May not	4.21	258
		P_4	Yes	0.28	33
		P_5	Yes	11.47	412
SMP	SMC	System	Yes	52.51	51
	MC	P_1	Yes	7.75	192
		P_2	Yes	1.11	50
		P_3	Yes	15.98	351
		P_4	Yes	0.29	33
		P_5	Yes	9.76	357

non-schedulability on the basis of counter-examples generated by UPPAAL to help refine the system configuration.

Fig. 20 shows the Gantt chart of a counter-example, where the task Tsk_2^3 in P_3 violates the constraint of the refresh period of Msg_2. The top two lines "partition" and "tscheduler" represent two scheduling-layer models `PartitionScheduler` and `AMPTaskScheduler`, respectively. For the line "partition", color red denotes the time outside P_3 and green is inside P_3. For "tscheduler", color red denotes that there are no tasks, color green denotes idle and the color blue denotes that the scheduler is occupied with a task. The next four lines from the top in Fig. 20 are task models, where a line is painted in green whenever a task stays at `Ready` state and in blue at `Running`. The communication-layer models correspond to the remaining lines of the Gantt chart. In these lines the transmitting of Msg_k are represented by the chart lines "msg$k$_snd", "vlink$k$_tx", "es$x$", and "switch_py", which denote the message-delivery delays of Msg_k through an AFDX network.

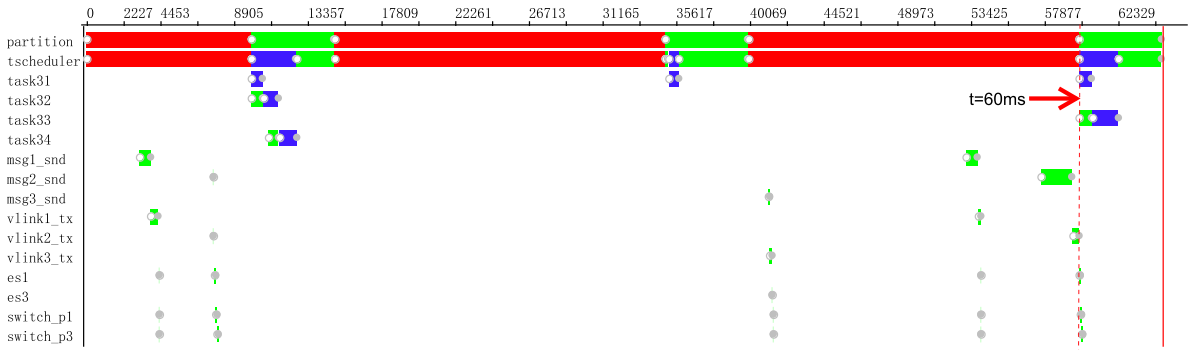


Fig. 20 Gantt chart of a counter-example (Times in microseconds)

The counter-example illustrates that network latency increases the risk of breaching the schedulability constraints.

Let t be the elapsed time since the initial instant $t_0 = 0$ shown in the Gantt chart. The first message of Msg_2 was sent by the message interface `msg2_snd` at $t = 7.625$ms, and reached the destination port at $t = 8.088$ms. When Tsk_2^3 was scheduled to read Msg_2 at $t = 60.000$ms, the age of the first received message indicated the value 51.912ms that had exceeded the refresh period. Thus, the copied message of Msg_2 was not a valid data sample. Although `msg2_snd` sent a new Msg_2 message at $t = 59.585$ms, the message did not arrive at the destination port until $t = 60.184$ms due to network latency.

B. Experiment 2 in SMP configuration

Considering the adverse effect of communication latency on schedulability, we adopt an SMP configuration instead of AMP in the module M_1, thereby making it possible to execute multiple tasks of the same partition simultaneously. Both of the processor cores *Core 0* and *Core 1* are assigned to the partitions P_1 and P_2 in M_1. The task set $\{Tsk_1^1, Tsk_3^1, Tsk_5^1, Tsk_2^2, Tsk_4^2\}$ has a processor core affinity *Core 0*, while the tasks in $\{Tsk_2^1, Tsk_4^1, Tsk_1^2, Tsk_3^2\}$ are assigned to an affinity for *Core 1*.

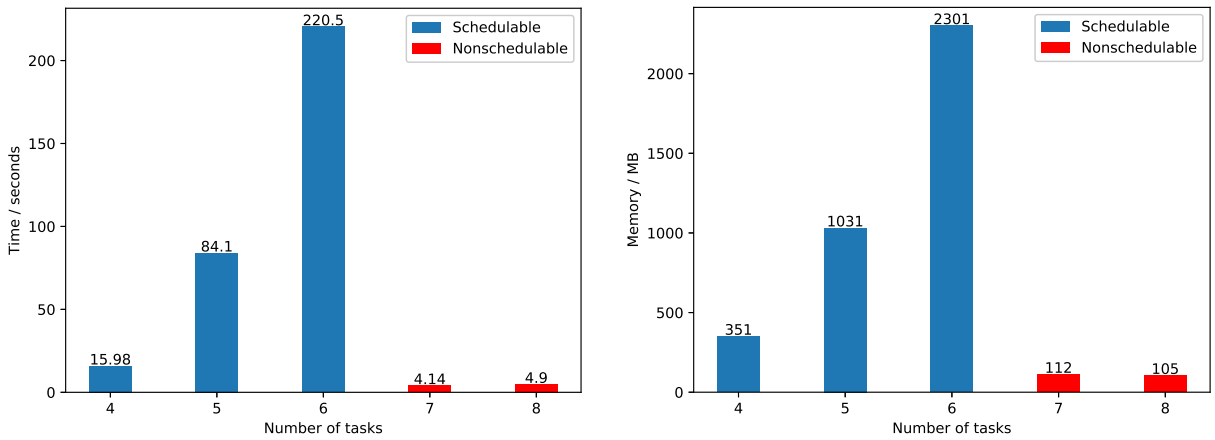
The schedulability analysis of the updated system was executed again. The result (The SMP case in Table 4) shows that the SMP configuration goes through the global SMC test and compositional verification of classical MC. Thus, the updated system finally achieves schedulability.

Moreover, the budgets of P_1 and P_2 can be further reduced in SMP configuration. We started an additional schedulability analysis from the above partition schedules and decreased the budgets of P_1 and P_2 iteratively with a step size $0.1ms$. The schedulability verification was repeated until the system had been verified as unschedulable. The refined partition schedules include five windows $\langle P_1, 0, 4.7\rangle$, $\langle P_2, 5, 3.2\rangle$, $\langle P_3, 10, 5\rangle$, $\langle P_4, 15, 5\rangle$, and $\langle P_5, 20, 5\rangle$, which spend the minimum budgets of P_1 and P_2.

Results of the Analysis

Table 4 shows the execution time and memory usage of the analysis tools in our two experiments. In compositional analysis (MC in Table 4), partition P_5 contains more instantiated models (18 processes) than the other four partitions. As a result, model-checking runs slower and requires more memory than the others. Nevertheless, the compositional analysis could be performed on ordinary personal computers within a very acceptable time.

Compared with the compositional way, global analysis based on the same Uppaal models would require 44 processes including all the 22 task models whose state space is much more complex than the others. This causes Uppaal classic MC to run out of memory within a few minutes, and thus makes the global analysis using classical MC infeasible. In contrast, SMC falsification testing can be quickly accomplished when we perform global analysis (SMC in Table 4), offering effective state space reduction.



(a) Time consumption

(b) Memory consumption

Fig. 21 Time and memory consumption in the performance experiment

C. Performance experiment

We argue that the main bottleneck of our method is the use of exact model-checking of the schedulability within a single partition, where the state-space is expected to grow exponentially with the number of concurrent components (e.g., tasks). To give an indication of this limit, this third experiment shows the performance and scalability of our proposed method by scaling up the number of tasks in a partition, thus also scaling up the state-space that has to be explored using model-checking as part of the compositional analysis. We keep the same configuration as experiment 2, but for each repetition we append one task from P_5 to partition P_3. Thus the compositional analysis was repeated four times as follows in addition to that of the original task sets:

- Compositional analysis where P_3 contains 5 tasks $\{Tsk_1^3, Tsk_2^3, Tsk_3^3, Tsk_4^3, Tsk_1^5\}$
- Compositional analysis where P_3 contains 6 tasks $\{Tsk_1^3, Tsk_2^3, Tsk_3^3, Tsk_4^3, Tsk_1^5, Tsk_2^5\}$
- Compositional analysis where P_3 contains 7 tasks $\{Tsk_1^3, Tsk_2^3, Tsk_3^3, Tsk_4^3, Tsk_1^5, Tsk_2^5, Tsk_3^5\}$
- Compositional analysis where P_3 contains 8 tasks $\{Tsk_1^3, Tsk_2^3, Tsk_3^3, Tsk_4^3, Tsk_1^5, Tsk_2^5, Tsk_3^5, Tsk_4^5\}$

All the tasks moved from P_5 to P_3 keep their original properties such as priority. Since P_3 and P_5 share the same end system, this migration does not affect the underlying network configuration.

As we scale up the size of the task set of P_3, the resource consumption for the verification grows exponentially.

Results of the Analysis

Figure 21 depicts the time (Fig. 21(a)) and memory (Fig. 21(b)) consumption of verification for P_3. Even though we only show three points on this graph both time and memory consumption seem to follow an exponential growth. At the beginning of the experiment, the verification of the original partition with 4 tasks is fast accomplished at a cost of 15.98s and 351MB, which can be satisfied easily by an ordinary personal computer. As we increase the number of tasks

within a partition, the consumption rises exponentially until this partition becomes non-schedulable. After P_3 contains more than 6 tasks, it turns non-schedulable and meanwhile its consumption will fall sharply. It is worth noticing that when a given partition is non-schedulable this will be quickly reported, which is important in a development situation.

We also conducted two experiments with 7 and 8 tasks where we also extended the partition budget of P_3 such that the partition would still be schedulable. For 7 tasks the verification took 637 seconds and used 6 GB of memory. For 8 tasks the verification used the full 12 GB of memory of the system on which the experiments were performed and did not complete.

As expected, the experiment reveals that the applicability of our compositional analysis largely depends on the number of tasks in each partition. This problem of state-space explosion is exactly the problem which the compositional verification technique is constructed to mitigate. Without the compositional method, model-based schedulability would be limited to a single partition, whereas we are able to address multiple partitions in a potentially distributed multi-core system.

On our ordinary personal computer[†], the compositional method can effectively cope with a schedulable system where there are up to 7 tasks in each partition. Moreover, we are able to handle a non-schedulable system with much larger partitions because the safety property of schedulability can be falsified faster than it can be confirmed by model checking.

VII. Discussion

In this section we will address the limitations of the analysis method outlined in this paper as well as discussing its strengths.

Certain features of the ARINC standard are not included in the current modeling reported in this paper. This includes:

- Hypervisor layer in Asymmetric Multi-Processing architecture, as described in Section III.
- Failure modes: (Section 2.4 *Health Monitor* in ARINC653P1-4[5])
- Vendor specific dynamic affinities: (Section 4.2.1 *O/S Multicore Implementation Compliance* in ARINC653P1-4[5])

Based on our understanding of these features, we see no hindrance that all three of these aspects could be modeled in our approach. We base this assessment on examples of which other types of systems that can be faithfully modeled using timed automata. In fact, we already have an AMP version of our models. The other two features would add to the complexity of the models, but we would not expect them to significantly add to the amount of time used by the tools to perform the analyses. In order to evaluate the safety of a platform it would be very important to include the failure modes of the system. Our method can easily handle having dynamic affinities where the individual task can

[†]MODEL: Lenovo Qitian M4500; CPU: Intel Core i5-4590@3.30GHz; MEM: 12GB RAM; OS: Ubuntu 16.04 LTS 64bit version.

be scheduled on a custom defined sub-set of cores. Adding these features to the modeling and analysis framework is future work.

In the current modelling, we assume that the effect on execution time of the shared underlying hardware resources such as private and shared caches, and busses are taken into account by the worst case execution time analysis.

One aspect that may need special attention if the modelling is to be extended with dynamic thread migration between the cores is that execution time may change abruptly when a thread is migrated to a different core as the thread might experience e.g. cold caches. It may not always be safe to assume that this situation can be absorbed in a WCET number. It may be possible to handle this situation at the schedulability analysis level, in our case by adding an extra amount of execution time when a task is migrated. However, this would need more details of the specific scheduling and migration policies of the specific operating system.

Our framework allows for very detailed modelling of task behaviors, in turn enabling a very refined and less conservative analysis under these specific assumptions. The detailed modelling is not mandated, so when the detailed information is not available, more abstract (thus more conservative) task behavior models can be used.

Our analysis is exact in the sense that if the modelled assumptions and supplied parameteres hold, then so does the computed outcome. If the assumptions are invalid, the results may also be invalid. Our main mechanism to deal with this is the ability to widen the specified parameters such as lower and upper bounds on execution time or message propagation delay. Similarly, it is possible to add more behaviors in the underlying scheduling system as non-deterministic choices. The analysis would then be exact under these widened assumptions. This will likely increase the computation time of the analysis as more behaviors need to be explored, and the results are also likely to be more conservative. So in our approach we can let the engineer decide on the level of details in the analysis.

We finally remark, that our present evaluation is based on a single case study, and future work is to determine if the successful results can be replicated on further cases. However, there are not many realistic (distributed, multi-core) ARINC workloads published.

VIII. Conclusion

The design of advanced avionics systems must rise to the challenge of multi-core processors and distributed architectures. We conclude that our UPPAAL-based approach presented in this paper is applicable to schedulability analysis of distributed avionics systems in both AMP and SMP multi-core configurations. We have also analyzed the scalability of our approach. As demonstrated by the case-study, we conclude that our approach is able to handle a realistic distributed multi-core system. The main limitation of scalability is the the number of tasks within a partition. We do not view this as a serious defect as it is often possible to separate tasks into multiple partitions without affecting the performance of the system.

The UPPAAL modeling framework is able to cover ARINC-653 multi-core computation as well as the AFDX network

environment. The combination of global SMC analysis and compositional analysis alleviates the state space explosion of classical model checking effectively. The case study shows better compatibility with legacy software in the AMP configuration and better parallel acceleration of applications in the SMP configuration. As future work, we plan to develop a model-based method of the automatic optimization of ARINC-653 scheduling systems, as well as conducting more realistic case-studies.

Appendix

The appendix consists of three sections. Appendix A contains the abbreviations used. Appendix B details the proofs of all the theorems in the paper. Appendix C presents the complete periodic task template in the UPPAAL modeling framework.

A. Abbreviations

Table 5 List of abbreviations

AFDX	Avionics Full Duplex Switched Etherne	pTPN	preemptive Time Petri Nets
AMP	Asymmetrical Multi-Processing	RDC	Remote Data Concentrators
BAG	Bandwidth Allocation Gap	SMC	Statistical Model Checking
BCET	Best Cast Execution Time	SMP	Symmetrical Multi-Processing
COTS	Commercial-of-the-Shelf	SWA	Stopwatch Automata
CPN	Colored Petri Nets	TA	Timed Automata
DIMA	Distributed Integrated Modular Avionics	TDM	Time Division Multiplexing
ES	End System	TIOTS	Timed I/O Transition Systems
FP	Fixed Priority	VL	Virtual Link
LHA	Linear Hybrid Automata	WCET	Worst Cast Execution Time
MC	Model Checking		

B. Theorem Proofs

Lemma 1 *Let $\mathcal{T}_i = \langle S_i, s_{i,0}, \Sigma_i, \rightarrow_i \rangle$, $i \in \{1,2\}$ be two TIOTSs. Assume that R is a timed selection simulation from $\mathcal{T}_1$ to $\mathcal{T}_2$. Then for all $(s_1, s_2) \in R$,*

1) *if $s_1 \overset{a?}{\Longrightarrow} s_1'$ for some $s_1' \in S_1$, $a \in \Sigma_2$, then there exists $s_2' \in S_2$ such that $s_2 \overset{a?}{\Longrightarrow} s_2'$ and $(s_1', s_2') \in R$*
2) *if $s_1 \overset{a!}{\Longrightarrow} s_1'$ for some $s_1' \in S_1$, $a \in \Sigma_2$, then there exists $s_2' \in S_2$ such that $s_2 \overset{a!}{\Longrightarrow} s_2'$ and $(s_1', s_2') \in R$*
3) *if $s_1 \overset{a}{\Longrightarrow} s_1'$ for some $s_1' \in S_1$, $a \in (\Sigma_1 \setminus \Sigma_2)$, then there exists $s_2' \in S_2$ such that $s_2 \overset{\mathbf{0}}{\Longrightarrow} s_2'$ and $(s_1', s_2') \in R$*
4) *if $s_1 \overset{\epsilon(d)}{\Longrightarrow} s_1'$ for some $s_1' \in S_1$, $d \geq 0$, then there exists $s_2' \in S_2$ such that $s_2 \overset{\epsilon(d)}{\Longrightarrow} s_2'$ and $(s_1', s_2') \in R$.*

Proof 1 *Consider $\mathcal{T}_1$, $\mathcal{T}_2$, s_1, s_2, and R in Lemma 1. From 3 of Definition 5 it is trivially the fact that if $s_1 \overset{\tau}{\rightarrow}^* s_1'$, $s_1' \in S_1$ then $s_2 \overset{\tau}{\rightarrow}^* s_2'$ for some $s_2' \in S_2$ such that $(s_1', s_2') \in R$. We denote this by $(*)$.*

Suppose $s_1 \overset{a?}{\Longrightarrow} s_1'$, $s_1' \in S_1$, and $a \in \Sigma_2$. Thus $s_1 \overset{\tau}{\rightarrow}^* s_1'' \overset{a?}{\longrightarrow} s_1''' \overset{\tau}{\rightarrow}^* s_1'$ for some $s_1'', s_1''' \in S_1$. From $(*)$ and 1 of Definition 5, we have that there exist $s_2', s_2'', s_2''' \in S_2$ such that $s_2 \overset{\tau}{\rightarrow}^* s_2'' \overset{a?}{\Longrightarrow} s_2''' \overset{\tau}{\rightarrow}^* s_2'$, i.e. $s_2 \overset{a?}{\Longrightarrow} s_2'$, where (s_1', s_2'),(s_1'', s_2''),$(s_1''', s_2''') \in R$. Hence 1 of Lemma 1 holds. Similarly 2 of Lemma 1 also holds.

Suppose $s_1 \overset{a}{\Longrightarrow} s_1'$ for some $s_1' \in S_1$, $a \in (\Sigma_1 \setminus \Sigma_2)$. Then $s_1 \overset{\tau}{\rightarrow}^* s_1'' \overset{a}{\rightarrow} s_1''' \overset{\tau}{\rightarrow}^* s_1'$ for some $s_1'', s_1''' \in S_1$. From $(*)$ and 3 of Definition 5, there exist $s_2', s_2'', s_2''' \in S_2$ such that $s_2 \overset{\tau}{\rightarrow}^* s_2'' \overset{\tau}{\rightarrow}^* s_2''' \overset{\tau}{\rightarrow}^* s_2'$ and (s_1', s_2'),(s_1'', s_2''),$(s_1''', s_2''') \in R$. Thus we have $s_2 \overset{\mathbf{0}}{\Longrightarrow} s_2'$ and 3 of Lemma 1 holds.

Finally, suppose $s_1 \stackrel{\epsilon(d)}{\Longrightarrow} s_1'$ *for some* $s_1' \in S_1$, $d \geq 0$. *First, if* $d = 0$ *then 4 of Lemma 1 holds because it is identical to* $(*)$. *Second, in the case of* $d > 0$ *we have* $s_1 \stackrel{\tau}{\rightarrow}^* s_1^{1\prime} \stackrel{\epsilon(d_1)}{\longrightarrow} s_1^{1\prime\prime} \stackrel{\tau}{\rightarrow}^* s_1^{2\prime} \stackrel{\epsilon(d_2)}{\longrightarrow} s_1^{2\prime\prime} \stackrel{\tau}{\rightarrow}^* \cdots \stackrel{\tau}{\rightarrow}^* s_1^{n\prime} \stackrel{\epsilon(d_n)}{\longrightarrow} s_1^{n\prime\prime} = s_1'$ *where* $\Sigma_{i=1}^n d_i = d$. *From From* $(*)$ *and 4 of Definition 5, there exist* $s_2^{1\prime}, s_2^{1\prime\prime}, s_2^{2\prime}, s_2^{2\prime\prime}, \ldots, s_2^{n\prime}, s_2^{n\prime\prime} \in S_2$ *such that* $s_2 \stackrel{\tau}{\rightarrow}^* s_2^{1\prime} \stackrel{\epsilon(d_1)}{\Longrightarrow} s_2^{1\prime\prime} \stackrel{\tau}{\rightarrow}^* s_2^{2\prime} \stackrel{\epsilon(d_2)}{\Longrightarrow} s_2^{2\prime\prime} \stackrel{\tau}{\rightarrow}^* \cdots \stackrel{\tau}{\rightarrow}^* s_2^{n\prime} \stackrel{\epsilon(d_n)}{\Longrightarrow} s_2^{n\prime\prime} = s_2'$ *and* $(s_1^{1\prime}, s_2^{1\prime})$, $(s_1^{1\prime\prime}, s_2^{1\prime\prime})$, $(s_1^{2\prime}, s_2^{2\prime})$, $\cdots$, $(s_1^{n\prime}, s_2^{n\prime})$, $(s_1^{n\prime\prime}, s_2^{n\prime\prime}) \in R$. *Hence we have* $s_2 \stackrel{\epsilon(d)}{\Longrightarrow} s_2'$ *and 4 of Lemma 1 holds.*

Lemma 2 *Let* $\mathcal{T}_i = \langle S_i, s_{i,0}, \Sigma_i, \rightarrow_i \rangle$, $i \in \{1, 2\}$ *be two compatible TIOTSs. Assume that* $\mathcal{T}_{1\|2} = \langle S_{1\|2}, s_{1\|2,0}, \Sigma_{1\|2}, \rightarrow_{1\|2} \rangle = \mathcal{T}_1 \| \mathcal{T}_2$. *Then for all* $\langle s_1, s_2 \rangle \in S_{1\|2}$,

1) *if* $s_1 \stackrel{a?}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{a?}{\Longrightarrow} s_2'$ *for some* $s_1' \in S_1$, $s_2' \in S_2$, $a \in \Sigma_1 \cap \Sigma_2$, *then there exists* $a \in \Sigma_{1\|2}$ *such that* $\langle s_1, s_2 \rangle \stackrel{a?}{\Longrightarrow} \langle s_1', s_2' \rangle$ *in* $\mathcal{T}_{1\|2}$
2) *if* $s_1 \stackrel{a!}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{a?}{\Longrightarrow} s_2'$, *or if* $s_1 \stackrel{a?}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{a!}{\Longrightarrow} s_2'$, *for some* $s_1' \in S_1$, $s_2' \in S_2$, $a \in \Sigma_1 \cap \Sigma_2$, *then there exists* $a \in \Sigma_{1\|2}$ *such that* $\langle s_1, s_2 \rangle \stackrel{a!}{\Longrightarrow} \langle s_1', s_2' \rangle$ *in* $\mathcal{T}_{1\|2}$
3) *if* $s_1 \stackrel{a}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{\mathbf{0}}{\Longrightarrow} s_2'$, *or if* $s_1 \stackrel{\mathbf{0}}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{a}{\Longrightarrow} s_2'$, *for some* $s_1' \in S_1$, $s_2' \in S_2$, $a \in \Sigma_1 \oplus \Sigma_2$, *then there exists* $a \in \Sigma_{1\|2}$ *such that* $\langle s_1, s_2 \rangle \stackrel{a}{\Longrightarrow} \langle s_1', s_2' \rangle$ *in* $\mathcal{T}_{1\|2}$
4) *if* $s_1 \stackrel{\epsilon(d)}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{\epsilon(d)}{\Longrightarrow} s_2'$ *for some* $s_1' \in S_1$, $s_2' \in S_2$, $d \geq 0$, *then there exists* $\langle s_1, s_2 \rangle \stackrel{\epsilon(d)}{\Longrightarrow} \langle s_1', s_2' \rangle$ *in* $\mathcal{T}_{1\|2}$.

Proof 2 *Consider* $\mathcal{T}_1$, $\mathcal{T}_2$, $s_1 \in S_1$, $s_2 \in S_2$, *and* $\langle s_1, s_2 \rangle \in S_{1\|2}$ *in Lemma 2. From the rules "INDEP-L" and "INDEP-R" it is trivially the fact that if* $s_1 \stackrel{\tau}{\rightarrow}^* s_1'$ *and* $s_2 \stackrel{\tau}{\rightarrow}^* s_2'$ *for some* $s_1' \in S_1$, $s_2' \in S_2$ *then* $\langle s_1, s_2 \rangle \stackrel{\tau}{\rightarrow}^* \langle s_1', s_2' \rangle$. *We denote this by* $(**)$.

Suppose $s_1 \stackrel{a?}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{a?}{\Longrightarrow} s_2'$ *for some* $s_1' \in S_1$, $s_2' \in S_2$, $a \in \Sigma_1 \cap \Sigma_2$. *Then there exist* s_1'', $s_1''' \in S_1$, s_2'', $s_2''' \in S_2$ *such that* $s_1 \stackrel{\tau}{\rightarrow}^* s_1'' \stackrel{a?}{\rightarrow} s_1''' \stackrel{\tau}{\rightarrow}^* s_1'$ *and* $s_2 \stackrel{\tau}{\rightarrow}^* s_2'' \stackrel{a?}{\rightarrow} s_2''' \stackrel{\tau}{\rightarrow}^* s_2'$. *From* $(**)$ *and the rule "SYNC-IN", we have* $\langle s_1, s_2 \rangle \stackrel{\tau}{\rightarrow}^* \langle s_1'', s_2'' \rangle \stackrel{a?}{\rightarrow} \langle s_1''', s_2''' \rangle \stackrel{\tau}{\rightarrow}^* \langle s_1', s_2' \rangle$. *Hence* $\langle s_1, s_2 \rangle \stackrel{a?}{\Longrightarrow} \langle s_1', s_2' \rangle$ *and 1 of Lemma 2 holds.*

Suppose $s_1 \stackrel{a!}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{a?}{\Longrightarrow} s_2'$ *for some* $s_1' \in S_1$, $s_2' \in S_2$, $a \in \Sigma_1 \cap \Sigma_2 \cap \Sigma^b$. *Then there exist* s_1'', $s_1''' \in S_1$, s_2'', $s_2''' \in S_2$ *such that* $s_1 \stackrel{\tau}{\rightarrow}^* s_1'' \stackrel{a!}{\rightarrow} s_1''' \stackrel{\tau}{\rightarrow}^* s_1'$ *and* $s_2 \stackrel{\tau}{\rightarrow}^* s_2'' \stackrel{a?}{\rightarrow} s_2''' \stackrel{\tau}{\rightarrow}^* s_2'$. *From* $(**)$ *and the rule "SYNC-IO", we have* $\langle s_1, s_2 \rangle \stackrel{\tau}{\rightarrow}^* \langle s_1'', s_2'' \rangle \stackrel{a!}{\rightarrow} \langle s_1''', s_2''' \rangle \stackrel{\tau}{\rightarrow}^* \langle s_1', s_2' \rangle$. *Hence* $\langle s_1, s_2 \rangle \stackrel{a!}{\Longrightarrow} \langle s_1', s_2' \rangle$. *Symmetrically we also have the same conclusion in the case of* $s_1 \stackrel{a?}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{a!}{\Longrightarrow} s_2'$. *Thus 2 of Lemma 2 holds.*

Suppose $s_1 \stackrel{a}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{\mathbf{0}}{\Longrightarrow} s_2'$ *for some* $s_1' \in S_1$, $s_2' \in S_2$, $a \in \Sigma_1 \oplus \Sigma_2$. *Then there exist* s_1'', $s_1''' \in S_1$, $s_2'' \in S_2$ *such that* $s_1 \stackrel{\tau}{\rightarrow}^* s_1'' \stackrel{a}{\rightarrow} s_1''' \stackrel{\tau}{\rightarrow}^* s_1'$ *and* $s_2 \stackrel{\tau}{\rightarrow}^* s_2'' \stackrel{\tau}{\rightarrow}^* s_2'$. *From* $(**)$ *and the rule "INDEP-L", we have* $\langle s_1, s_2 \rangle \stackrel{\tau}{\rightarrow}^* \langle s_1'', s_2'' \rangle \stackrel{a}{\rightarrow} \langle s_1''', s_2'' \rangle \stackrel{\tau}{\rightarrow}^* \langle s_1', s_2' \rangle$. *Hence* $\langle s_1, s_2 \rangle \stackrel{a}{\Longrightarrow} \langle s_1', s_2' \rangle$. *Symmetrically we also have the same conclusion in the case of* $s_1 \stackrel{\mathbf{0}}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{a}{\Longrightarrow} s_2'$. *Thus 3 of Lemma 2 holds.*

Suppose $s_1 \stackrel{\epsilon(d)}{\Longrightarrow} s_1'$ *and* $s_2 \stackrel{\epsilon(d)}{\Longrightarrow} s_2'$ *for some* $s_1' \in S_1$, $s_2' \in S_2$, $d \geq 0$. *From* $(**)$ *we have that 4 of Lemma 2 holds in the case of* $d = 0$. *Consider the case of* $d \geq 0$. $s_1 \stackrel{\epsilon(d)}{\Longrightarrow} s_1'$ *is equivalent to* $s_1 \stackrel{\tau}{\rightarrow}^* s_1^{1\prime} \stackrel{\epsilon(d_1)}{\longrightarrow} s_1^{1\prime\prime} \stackrel{\tau}{\rightarrow}^* s_1^{2\prime} \stackrel{\epsilon(d_2)}{\longrightarrow} s_1^{2\prime\prime} \stackrel{\tau}{\rightarrow}^* \cdots \stackrel{\tau}{\rightarrow}^* s_1^{n\prime} \stackrel{\epsilon(d_n)}{\longrightarrow} s_1^{n\prime\prime} \stackrel{\tau}{\rightarrow}^* s_1'$ *where* $n \in \mathbf{N}_+, \Sigma_{i=1}^n d_i = d$. *We now prove 4 of Lemma 2 using mathematical induction.*

Assume that $s_2 \overset{\epsilon(d)}{\Longrightarrow} s_2'$ *contains a transition chain* $s_2 \overset{\tau}{\rightarrow}^* s_2^{1\prime} \overset{\epsilon(d_1')}{\longrightarrow} s_2^{1\prime\prime} \overset{\tau}{\rightarrow}^* s_2^{2\prime} \overset{\epsilon(d_2')}{\longrightarrow} s_2^{2\prime\prime} \overset{\tau}{\rightarrow}^* \cdots \overset{\tau}{\rightarrow}^* s_2^{m\prime} \overset{\epsilon(d_m')}{\longrightarrow} s_2^{m\prime\prime} \overset{\tau}{\rightarrow}^* s_2'$ *where* $m \in \mathbf{N}_+, \Sigma_{i=1}^m d_i' = d$.

If $n = 1$ *then* $s_1 \overset{\epsilon(d)}{\Longrightarrow} s_1'$ *will be equivalent to* $s_1 \overset{\tau}{\rightarrow}^* s_1^{1\prime} \overset{\epsilon(d)}{\longrightarrow} s_1^{1\prime\prime} \overset{\tau}{\rightarrow}^* s_1'$. *From time additivity of TIOTS, there exist* $s_1^{2\prime}, s_1^{3\prime}, \ldots, s_1^{m\prime} \in S_1$ *such that* $s_1 \overset{\tau}{\rightarrow}^* s_1^{1\prime} \overset{\epsilon(d_1')}{\longrightarrow} s_1^{2\prime} \overset{\epsilon(d_2')}{\longrightarrow} \cdots \overset{\epsilon(d_m')}{\longrightarrow} s_1^{1\prime\prime} \overset{\tau}{\rightarrow}^* s_1'$. *By* $(**)$ *and the rule "DELAY", we have the transition chain* $\langle s_1, s_2\rangle \overset{\tau}{\rightarrow}^* \langle s_1^{1\prime}, s_2^{1\prime}\rangle \overset{\epsilon(d_1')}{\longrightarrow} \langle s_1^{2\prime}, s_2^{1\prime\prime}\rangle \overset{\tau}{\rightarrow}^* \langle s_1^{2\prime}, s_2^{2\prime}\rangle \overset{\epsilon(d_2')}{\longrightarrow} \langle s_1^{3\prime}, s_2^{2\prime\prime}\rangle \overset{\tau}{\rightarrow}^* \cdots \overset{\tau}{\rightarrow}^* \langle s_1^{m\prime}, s_2^{m\prime}\rangle \overset{\epsilon(d_m')}{\longrightarrow} \langle s_1^{1\prime\prime}, s_2^{m\prime\prime}\rangle \overset{\tau}{\rightarrow}^* \langle s_1', s_2'\rangle$. *Thus there exists* $\langle s_1, s_2\rangle \overset{\epsilon(d)}{\Longrightarrow} \langle s_1', s_2'\rangle$ *in* $\mathcal{T}_{1\|2}$.

We assume that there exists $\langle s_1, s_2\rangle \overset{\epsilon(d)}{\Longrightarrow} \langle s_1', s_2'\rangle$ *in* $\mathcal{T}_{1\|2}$ *if* $n = t, t \in \mathbf{N}_+$. *If* $n = t+1$ *then* $s_1 \overset{\epsilon(d)}{\Longrightarrow} s_1'$ *should contain a transition chain* $s_1 \overset{\tau}{\rightarrow}^* s_1^{1\prime} \overset{\epsilon(d_1)}{\longrightarrow} s_1^{1\prime\prime} \overset{\tau}{\rightarrow}^* s_1^{2\prime} \overset{\epsilon(d_2)}{\longrightarrow} s_1^{2\prime\prime} \overset{\tau}{\rightarrow}^* \cdots \overset{\tau}{\rightarrow}^* s_1^{t\prime} \overset{\epsilon(d_t)}{\longrightarrow} s_1^{t\prime\prime} \overset{\tau}{\rightarrow}^* s_1^{(t+1)\prime} \overset{\epsilon(d_{t+1})}{\longrightarrow} s_1^{(t+1)\prime\prime} \overset{\tau}{\rightarrow}^* s_1'$. *Let* $d' = d - d_{t+1}$. *From time additivity of TIOTS, there exists* $s_2^r \in S_2$ *such that* $s_2 \overset{\epsilon(d')}{\Longrightarrow} s_2^r \overset{\epsilon(d_{t+1})}{\Longrightarrow} s_2'$. *By the assumption in the case of* $n = t$, *we have that* $\langle s_1, s_2\rangle \overset{\epsilon(d')}{\Longrightarrow} \langle s_1^{t\prime\prime}, s_2^r\rangle$. *Consider the transitions* $s_1^{t\prime\prime} \overset{\tau}{\rightarrow}^* s_1^{(t+1)\prime} \overset{\epsilon(d_{t+1})}{\longrightarrow} s_1^{(t+1)\prime\prime} \overset{\tau}{\rightarrow}^* s_1'$ *and* $s_2^r \overset{\epsilon(d_{t+1})}{\Longrightarrow} s_2'$. *From the conclusion under the assumption* $n = 1$, $\langle s_1^{t\prime\prime}, s_2^r\rangle \overset{\epsilon(d_{t+1})}{\Longrightarrow} \langle s_1', s_2'\rangle$ *exists in* $\mathcal{T}_{1\|2}$ *and we also have* $\langle s_1, s_2\rangle \overset{\epsilon(d)}{\Longrightarrow} \langle s_1', s_2'\rangle$ *in the case* $n = t + 1$. *Hence 4 of Lemma 2 holds.*

Proof 3 (Proof of Theorem 1) *A preorder should be reflexive and transitive. For any TIOTS* $\mathcal{T} = \langle S, s_0, \Sigma, \rightarrow\rangle$, *the binary relation* $R = \{(s, s) | s \in S\}$ *trivially conforming to Definition 5 is a timed selection simulation from* $\mathcal{T}$ *to* $\mathcal{T}$, *i.e.* $\mathcal{T} \preceq \mathcal{T}$. *Hence reflexivity holds.*

We now show the transitivity of timed selection simulation. Consider any three TIOTSs $\mathcal{T}_i = \langle S_i, s_{i,0}, \Sigma_i, \rightarrow_i\rangle$, $i \in \{1, 2, 3\}$. *Assume that* R_1 *is a timed selection simulation from* $\mathcal{T}_1$ *to* $\mathcal{T}_2$ *and* R_2 *a timed selection simulation from* $\mathcal{T}_2$ *to* $\mathcal{T}_3$. *We prove that the new relation* $R = R_1R_2$ *is a timed selection simulation from* $\mathcal{T}_1$ *to* $\mathcal{T}_3$.

From Definition 5 we have $(s_{0,1}, s_{0,2}) \in R_1$ *and* $(s_{0,2}, s_{0,3}) \in R_2$. *Hence* $(s_{0,1}, s_{0,3}) \in R$. *For any* $(s_1, s_3) \in R$, *there exists* $s_2 \in S_2$ *such that* $(s_1, s_2) \in R_1$ *and* $(s_2, s_3) \in R_2$. *By Definition 5,* $g(s_1) = g(s_2)$ *and* $g(s_2) = g(s_3)$. *Thus* $g(s_1) = g(s_3)$. *Consider the four conditions of Definition 5.*

Suppose $s_1 \overset{a?}{\longrightarrow} s_1'$, $s_1' \in S_1$, *and* $a \in \Sigma_3$. *Since* $\mathcal{T}_2 \preceq \mathcal{T}_3$, *we have* $\Sigma_3 \subseteq \Sigma_2$ *and thus* $a \in \Sigma_2$. *Since* $\mathcal{T}_1 \preceq \mathcal{T}_2$, *there exists* $s_2' \in S_2$ *such that* $s_2 \overset{a?}{\Longrightarrow} s_2'$ *and* $(s_1', s_2') \in R_1$. *Since* $\mathcal{T}_2 \preceq \mathcal{T}_3$, *by 1 of Lemma 1 there exists* $s_3' \in S_3$ *such that* $s_3 \overset{a?}{\Longrightarrow} s_3'$ *and* $(s_2', s_3') \in R_2$. *Thus* $(s_1', s_3') \in R$ *and condition 1 of Definition 5 holds. Similarly condition 2 of Definition 5 also holds.*

Suppose $s_1 \overset{a}{\rightarrow} s_1'$, $s_1' \in S_1$, *and* $a \in (\Sigma_1 \setminus \Sigma_3)$. *If* $a \in \Sigma_2$, *then* $s_2 \overset{a}{\Longrightarrow} s_2', s_2' \in S_2$ *and* $(s_1', s_2') \in R_1$ *for* $\mathcal{T}_1 \preceq \mathcal{T}_2$ *and thus* $s_3 \overset{\mathbf{0}}{\Longrightarrow} s_3', s_3' \in S_3$ *and* $(s_2', s_3') \in R_2$ *for* $\mathcal{T}_2 \preceq \mathcal{T}_3$. *Hence* $(s_1', s_3') \in R$ *in the case of* $a \in \Sigma_2$. *If* $a \notin \Sigma_2$, *then* $s_2 \overset{\mathbf{0}}{\Longrightarrow} s_2', s_2' \in S_2$ *and* $(s_1', s_2') \in R_1$ *for* $\mathcal{T}_1 \preceq \mathcal{T}_2$. *From Lemma 1 and* $\mathcal{T}_2 \preceq \mathcal{T}_3$, *we have* $s_3 \overset{\mathbf{0}}{\Longrightarrow} s_3', s_3' \in S_3$ *and* $(s_2', s_3') \in R_2$. *Thus* $(s_1', s_3') \in R$ *in this case.*

Suppose $s_1 \overset{\epsilon(d)}{\longrightarrow} s_1'$, $s_1' \in S_1$, *and* $d \geq 0$. *From* $\mathcal{T}_1 \preceq \mathcal{T}_2$, $s_2 \overset{\epsilon(d)}{\Longrightarrow} s_2', s_2' \in S_2$ *and* $(s_1', s_2') \in R_1$. *From Lemma 1 and* $\mathcal{T}_2 \preceq \mathcal{T}_3$, *we have* $s_3 \overset{\epsilon(d)}{\Longrightarrow} s_3', s_3' \in S_3$ *and* $(s_2', s_3') \in R_2$. *Thus* $(s_1', s_3') \in R$ *and both condition 3 and 4 of Definition 5 hold.*

Therefore, R is a timed selection simulation from $\mathcal{T}_1$ to $\mathcal{T}_3$, and transitivity of timed selection simulation holds.

Proof 4 (Proof of Theorem 2) *Let S_i be the state set of $\mathcal{T}_i$. Let R be a timed selection simulation from $\mathcal{T}_1$ to $\mathcal{T}_2$. Note that $\mathcal{T}_i \models \varphi$ iff for any reachable state $s_i \in S_i$ $g(s_i) = false$. We denote this by $(*)$.*

From Definition 5 and $\mathcal{T}_1 \preceq \mathcal{T}_2$ we have that for each reachable state $s_1 \in S_1$, there exists a reachable state $s_2 \in S_2$ such that $(s_1, s_2) \in R$ and $g(s_1) = g(s_2)$. Since $\mathcal{T}_2 \models \varphi$ and $(*)$, $g(s_2) = false$ for each reachable state $s_2 \in S_2$. Thus $g(s_1) = false$ for any reachable state $s_1 \in S_1$. From $(*)$, we have $\mathcal{T}_1 \models \varphi$.

Proof 5 (Proof of Theorem 3) *Let S_i be the state set of $\mathcal{T}_i$. Assume that R_1 and R_2 are timed selection simulations from $\mathcal{T}_1$ to $\mathcal{T}_2$ and from $\mathcal{T}_1$ to $\mathcal{T}_3$, respectively. Let R be a binary relation from S_1 to $S_2 \times S_3$ such that $(s_1, \langle s_2, s_3 \rangle) \in R$ iff $(s_1, s_2) \in R_1$ and $(s_1, s_3) \in R_2$ for any $s_1 \in S_1, s_2 \in S_2, s_3 \in S_3$. We now prove R is a timed selection simulation relation.*

Suppose $s_{i,0}$ is the initial state of $\mathcal{T}_i$. By assumption we have $(s_{1,0}, s_{2,0}) \in R_1$ and $(s_{1,0}, s_{3,0}) \in R_2$. Thus $(s_{1,0}, \langle s_{2,0}, s_{3,0} \rangle) \in R$ from the definition of R.

Whenever $(s_1, s_2) \in R_1$ and $(s_1, s_3) \in R_2$, $g(s_1) = g(s_2)$ and $g(s_1) = g(s_3)$ will hold. Hence, from the definition of the function g, we have $g(s_1) = g(\langle s_2, s_3 \rangle)$ for any $(s_1, \langle s_2, s_3 \rangle) \in R$.

Let Σ_i be the action set of $\mathcal{T}_i$. Let I_i and O_i be the input and output action set in Σ_i respectively. From Definition 4, for any compositional TIOTS $\mathcal{T}_2 \| \mathcal{T}_3$ we have $\Sigma_{2\|3} = I_{2\|3} \oplus O_{2\|3}$, $I_{2\|3} = (I_2 \setminus O_3) \cup (I_3 \setminus O_2)$, and $O_{2\|3} = O_2 \cup O_3$. Since $\Sigma_2 \subseteq \Sigma_1$, $\Sigma_3 \subseteq \Sigma_1$ and $\mathcal{T}_2$ and $\mathcal{T}_3$ are compatible, we have

$$
\begin{aligned}
&\Sigma_2 \cup \Sigma_3 \\
=&(I_2 \oplus O_2) \cup (I_3 \oplus O_3) \\
=&[(I_2 \cup O_2) \setminus (I_2 \cap O_2)] \cup [(I_3 \cup O_3) \setminus (I_3 \cap O_3)] \\
=&(I_2 \cup O_2 \cup I_3 \cup O_3) \setminus [(I_3 \cap O_3) \setminus (I_2 \cup O_2)] \setminus [(I_2 \cap O_2) \setminus (I_3 \cup O_3)] \\
\subseteq&\Sigma_1
\end{aligned}
\tag{26}
$$

Let $I_2' = I_2 \setminus O_3$ *and* $I_3' = I_3 \setminus O_2$.

$$
\begin{aligned}
&\Sigma_{2\|3} \\
=&(I_2' \cup I_3') \oplus (O_2 \cup O_3) \\
=&(I_2' \cup O_2 \cup I_3' \cup O_3) \setminus [(I_2' \cup I_3') \cap (O_2 \cup O_3)] \\
=&(I_2' \cup O_2 \cup I_3' \cup O_3) \setminus (I_2' \cap O_2) \setminus (I_3' \cap O_3) \setminus (I_2' \cap O_3) \setminus (I_3' \cap O_2) \\
=&(I_2 \cup O_2 \cup I_3 \cup O_3) \setminus (I_2 \cap O_2) \setminus (I_3 \cap O_3) \\
\subseteq&(I_2 \cup O_2 \cup I_3 \cup O_3) \setminus [(I_2 \cap O_2) \setminus (I_3 \cup O_3)] \setminus [(I_3 \cap O_3) \setminus (I_2 \cup O_2)] \\
=&\Sigma_2 \cup \Sigma_3
\end{aligned}
\tag{27}
$$

Thus $\Sigma_{2\|3} \subseteq \Sigma_1$.

Assume $(s_1, s_2) \in R_1$ *and* $(s_1, s_3) \in R_2$ *for some* $s_1 \in S_1, s_2 \in S_2, s_3 \in S_3$. *Then* $(s_1, \langle s_2, s_3 \rangle) \in R$. *Consider each of the conditions in Definition 5.*

Suppose $s_1 \xrightarrow{a?} s_1'$ *for some* $a \in \Sigma_{2\|3}$. *Thus* $a \in \Sigma_2 \cup \Sigma_3$. *There are the following two cases:*

Case 1: $a \in \Sigma_2 \cap \Sigma_3$. *By simulation definition we have* $s_2 \overset{a?}{\Longrightarrow} s_2'$ *and* $s_3 \overset{a?}{\Longrightarrow} s_3'$ *for some* $s_2' \in S_2, s_3' \in S_3$ *such that* $(s_1', s_2') \in R_1$ *and* $(s_1', s_3') \in R_2$. *Hence* $(s_1', \langle s_2', s_3' \rangle) \in R$, *and from 1 of Lemma 2 there exists* $\langle s_2, s_3 \rangle \overset{a?}{\Longrightarrow} \langle s_2', s_3' \rangle$ *in* $\mathcal{T}_2 \| \mathcal{T}_3$.

Case 2: $a \in \Sigma_2 \oplus \Sigma_3$. *By simulation definition we have that* $s_2 \overset{a?}{\Longrightarrow} s_2'$ *and* $s_3 \overset{\mathbf{0}}{\Longrightarrow} s_3'$, *or* $s_2 \overset{\mathbf{0}}{\Longrightarrow} s_2'$ *and* $s_3 \overset{a?}{\Longrightarrow} s_3'$, *for some* $s_2' \in S_2, s_3' \in S_3$ *such that* $(s_1', s_2') \in R_1$ *and* $(s_1', s_3') \in R_2$. *Hence* $(s_1', \langle s_2', s_3' \rangle) \in R$, *and from 3 of Lemma 2 there exists* $\langle s_2, s_3 \rangle \overset{a?}{\Longrightarrow} \langle s_2', s_3' \rangle$ *in* $\mathcal{T}_2 \| \mathcal{T}_3$.

Suppose $s_1 \xrightarrow{a!} s_1'$ *for some* $a \in \Sigma_{2\|3}$. *There are also two cases:*

Case 1: $a \in \Sigma_2 \cap \Sigma_3$. *By simulation definition we have that* $s_2 \overset{a!}{\Longrightarrow} s_2'$ *and* $s_3 \overset{a?}{\Longrightarrow} s_3'$, *or* $s_2 \overset{a?}{\Longrightarrow} s_2'$ *and* $s_3 \overset{a!}{\Longrightarrow} s_3'$, *for some* $s_2' \in S_2, s_3' \in S_3$ *such that* $(s_1', s_2') \in R_1$ *and* $(s_1', s_3') \in R_2$. *Hence* $(s_1', \langle s_2', s_3' \rangle) \in R$, *and from 2 of Lemma 2 there exists* $\langle s_2, s_3 \rangle \overset{a!}{\Longrightarrow} \langle s_2', s_3' \rangle$ *in* $\mathcal{T}_2 \| \mathcal{T}_3$.

Case 2: $a \in \Sigma_2 \oplus \Sigma_3$. *By simulation definition we have that* $s_2 \overset{a!}{\Longrightarrow} s_2'$ *and* $s_3 \overset{\mathbf{0}}{\Longrightarrow} s_3'$, *or* $s_2 \overset{\mathbf{0}}{\Longrightarrow} s_2'$ *and* $s_3 \overset{a!}{\Longrightarrow} s_3'$, *for some* $s_2' \in S_2, s_3' \in S_3$ *such that* $(s_1', s_2') \in R_1$ *and* $(s_1', s_3') \in R_2$. *Hence* $(s_1', \langle s_2', s_3' \rangle) \in R$, *and from 3 of Lemma 2 there exists* $\langle s_2, s_3 \rangle \overset{a!}{\Longrightarrow} \langle s_2', s_3' \rangle$ *in* $\mathcal{T}_2 \| \mathcal{T}_3$.

Suppose $s_1 \xrightarrow{a} s_1'$ *for some* $a \in \Sigma_1 \setminus \Sigma_{2\|3}$. *Since* $\Sigma_{2\|3} \subseteq (\Sigma_2 \cup \Sigma_3) \subseteq \Sigma_1$, *there are the following two cases:*

Case 1: $a \in \Sigma_1 \setminus (\Sigma_2 \cup \Sigma_3)$. *Thus* $a \in \Sigma_1 \setminus \Sigma_2$ *and* $a \in \Sigma_1 \setminus \Sigma_3$. *By simulation definition we have that* $s_2 \overset{\mathbf{0}}{\Longrightarrow} s_2'$ *and* $s_3 \overset{\mathbf{0}}{\Longrightarrow} s_3'$ *for some* $s_2' \in S_2, s_3' \in S_3$ *such that* $(s_1', s_2') \in R_1$ *and* $(s_1', s_3') \in R_2$. *Hence* $(s_1', \langle s_2', s_3' \rangle) \in R$, *and from 4 of Lemma 2 there exists* $\langle s_2, s_3 \rangle \overset{\mathbf{0}}{\Longrightarrow} \langle s_2', s_3' \rangle$ *in* $\mathcal{T}_2 \| \mathcal{T}_3$.

Case 2: $a \in (\Sigma_2 \cup \Sigma_3) \setminus \Sigma_{2\|3}$. *We obtain* $(\Sigma_2 \cup \Sigma_3) \setminus \Sigma_{2\|3} \subseteq (I_2 \cap O_2) \cup (I_3 \cap O_3)$ *from Eq. (27). Thus there is no*

transition with the action a according to the definition of TIOTS. However, from $\mathcal{T}_1 \preceq \mathcal{T}_2, \mathcal{T}_1 \preceq \mathcal{T}_3$ *we have that* $s_2 \overset{a}{\Longrightarrow} s_2'$ *and* $s_3 \overset{a}{\Longrightarrow} s_3'$ *for some* $s_2' \in S_2, s_3' \in S_3$*, which contradicts the fact that* $a \in (\Sigma_2 \cup \Sigma_3) \setminus \Sigma_{2\|3}$*. Hence such an action a does not exist in this case.*

Suppose $s_1 \xrightarrow{\epsilon(d)} s_1'$ *and* $d \geq 0$*. By simulation definition we have that* $s_2 \overset{\epsilon(d)}{\Longrightarrow} s_2'$ *and* $s_3 \overset{\epsilon(d)}{\Longrightarrow} s_3'$ *for some* $s_2' \in S_2, s_3' \in S_3$ *such that* $(s_1', s_2') \in R_1$ *and* $(s_1', s_3') \in R_2$*. Hence* $(s_1', \langle s_2', s_3' \rangle) \in R$*, and from 4 of Lemma 2 there exists* $\langle s_2, s_3 \rangle \overset{\epsilon(d)}{\Longrightarrow} \langle s_2', s_3' \rangle$ *in* $\mathcal{T}_2 \| \mathcal{T}_3$*. All the conditions hold and thus* $\mathcal{T}_1 \preceq \mathcal{T}_2 \| \mathcal{T}_3$*.*

Proof 6 (Proof of Theorem 4) *Let* S_i *be the state set of* $\mathcal{T}_i$*. Assume that* R_1 *and* R_2 *are timed selection simulations from* $\mathcal{T}_1$ *to* $\mathcal{T}_2$ *and from* $\mathcal{T}_3$ *to* $\mathcal{T}_4$*, respectively. Let R be a binary relation from* $S_1 \times S_3$ *to* $S_2 \times S_4$ *such that* $(\langle s_1, s_3 \rangle, \langle s_2, s_4 \rangle) \in R$ *iff* $(s_1, s_2) \in R_1$ *and* $(s_3, s_4) \in R_2$ *for any* $s_1 \in S_1, s_2 \in S_2, s_3 \in S_3, s_4 \in S_4$*. We now prove R is a timed selection simulation relation.*

Suppose $s_{i,0}$ *is the initial state of* $\mathcal{T}_i$*. By assumption (1) we have* $(s_{1,0}, s_{2,0}) \in R_1$ *and* $(s_{3,0}, s_{4,0}) \in R_2$*. Thus* $(\langle s_{1,0}, s_{3,0} \rangle, \langle s_{2,0}, s_{4,0} \rangle) \in R$ *from the definition of R.*

Whenever $(s_1, s_2) \in R_1$ *and* $(s_3, s_4) \in R_2$*,* $g(s_1) = g(s_2)$ *and* $g(s_3) = g(s_4)$ *will hold. Hence, from the definition of the function g, we have* $g(\langle s_1, s_3 \rangle) = g(\langle s_2, s_4 \rangle)$ *for any* $(\langle s_1, s_3 \rangle, \langle s_2, s_4 \rangle) \in R$*.*

Let Σ_i *be the action set of* $\mathcal{T}_i$*. Let* I_i *and* O_i *be the input and output action set in* Σ_i *respectively. From Definition 4, for any compositional TIOTS* $\mathcal{T}_i \| \mathcal{T}_j$ *we have* $\Sigma_{i\|j} = I_{i\|j} \oplus O_{i\|j}$*,* $I_{i\|j} = (I_i \setminus O_j) \cup (I_j \setminus O_i)$*, and* $O_{i\|j} = O_i \cup O_j$*. Let* $I_i' = I_i \setminus O_j$ *and* $I_j' = I_j \setminus O_i$*. By assumption (1) and Definition 5 we have* $\Sigma_2 \subseteq \Sigma_1$ *and* $\Sigma_4 \subseteq \Sigma_3$*. Then* $\Sigma_2 \cup \Sigma_4 \subseteq \Sigma_1 \cup \Sigma_3$*. We now prove* $\Sigma_{2\|4} \subseteq \Sigma_{1\|3}$*.*

Assume for the sake of contradiction that there exists $b \in \Sigma_{2\|4}$ *but* $b \notin \Sigma_{1\|3}$*.*

$$
\begin{aligned}
&\Sigma_{1\|3} \\
=&(I_1 \cup O_1 \cup I_3 \cup O_3) \setminus (I_1 \cap O_1) \setminus (I_3 \cap O_3) && \text{by Eq. (27)} \\
=&(I_1 \cup O_1 \cup I_3 \cup O_3) && \text{by Definition 2} \\
\subseteq&\Sigma_1 \cup \Sigma_3 && \text{by Eq. (27)} \\
=&(I_1 \cup O_1 \cup I_3 \cup O_3) \setminus [(I_3 \cap O_3) \setminus (I_1 \cup O_1)] \setminus [(I_1 \cap O_1) \setminus (I_3 \cup O_3)] && \text{by Eq. (26)} \\
=&(I_1 \cup O_1 \cup I_3 \cup O_3) && \text{by Definition 2}
\end{aligned} \tag{28}
$$

Thus $\Sigma_{1\|3} = \Sigma_1 \cup \Sigma_3$*. Similarly,* $\Sigma_{2\|4} = \Sigma_2 \cup \Sigma_4$*. From* $b \in \Sigma_{2\|4}$ *and* $\Sigma_2 \cup \Sigma_4 \subseteq \Sigma_1 \cup \Sigma_3$*, we obtain* $b \in \Sigma_1 \cup \Sigma_3$*, which contradicts the assumption* $b \notin \Sigma_{1\|3}$*. Thus* $b \in \Sigma_{2\|4}$ *implies* $b \in \Sigma_{1\|3}$*, and we have that* $\Sigma_{2\|4} \subseteq \Sigma_{1\|3}$*.*

Assume $(s_1, s_2) \in R_1$ *and* $(s_3, s_4) \in R_2$ *for some* $s_1 \in S_1, s_2 \in S_2, s_3 \in S_3, s_4 \in S_4$*. Then* $(\langle s_1, s_3 \rangle, \langle s_2, s_4 \rangle) \in R$*. Consider each of the conditions in Definition 5.*

Suppose $\langle s_1, s_3 \rangle \xrightarrow{a?} \langle s_1', s_3' \rangle$ *for some* $a \in \Sigma_{2\|4}$*. Thus* $a \in \Sigma_2 \cup \Sigma_4$*. There are the following two cases:*

Table 6 Transition Set 1

Premises	No.	$\mathcal{T}_1$			$\mathcal{T}_2$			$\mathcal{T}_3$			$\mathcal{T}_4$		
		st_1	Tr_1	st_1'	st_2	Tr_2	st_2'	st_3	Tr_3	st_3'	st_4	Tr_4	st_4'
$a \in \Sigma_2 \setminus \Sigma_4$	1	s_1	$\xrightarrow{a?}$	s_1'	s_2	$\Longrightarrow^{a?}$	s_2'	s_3	$\xrightarrow{a!}$	s_3'	s_4	$\Longrightarrow^{\mathbf{0}}$	s_4'
	2	s_1	$\xrightarrow{a!}$	s_1'	s_2	$\Longrightarrow^{a!}$	s_2'	s_3	$\xrightarrow{a?}$	s_3'	s_4	$\Longrightarrow^{\mathbf{0}}$	s_4'
	3	s_1	$\xrightarrow{a!}$	s_1'	s_2	$\Longrightarrow^{a!}$	s_2'	s_3	$\xrightarrow{\mathbf{0}}$	s_3	s_4	$\xrightarrow{\mathbf{0}}$	s_4
$a \in \Sigma_4 \setminus \Sigma_2$	4	s_1	$\xrightarrow{a?}$	s_1'	s_2	$\Longrightarrow^{\mathbf{0}}$	s_2'	s_3	$\xrightarrow{a!}$	s_3'	s_4	$\Longrightarrow^{a!}$	s_4'
	5	s_1	$\xrightarrow{a!}$	s_1'	s_2	$\Longrightarrow^{\mathbf{0}}$	s_2'	s_3	$\xrightarrow{a?}$	s_3'	s_4	$\Longrightarrow^{a?}$	s_4'
	6	s_1	$\xrightarrow{\mathbf{0}}$	s_1	s_2	$\xrightarrow{\mathbf{0}}$	s_2	s_3	$\xrightarrow{a!}$	s_3'	s_4	$\Longrightarrow^{a!}$	s_4'

Case 1: $a \in \Sigma_2 \cap \Sigma_4$. *Since* $\Sigma_2 \subseteq \Sigma_1$ *and* $\Sigma_4 \subseteq \Sigma_3$, $a \in \Sigma_1 \cap \Sigma_3$. *According to the rule "SYNC-IN",* $s_1 \xrightarrow{a?} s_1'$ *and* $s_3 \xrightarrow{a?} s_3'$. *By simulation definition we have* $s_2 \Longrightarrow^{a?} s_2'$ *and* $s_4 \Longrightarrow^{a?} s_4'$ *for some* $s_2' \in S_2, s_4' \in S_4$ *such that* $(s_1', s_2') \in R_1$ *and* $(s_3', s_4') \in R_2$. *Hence* $(\langle s_1', s_3'\rangle, \langle s_2', s_4'\rangle) \in R$, *and from 1 of Lemma 2 there exists* $\langle s_2, s_4\rangle \Longrightarrow^{a?} \langle s_2', s_4'\rangle$ *in* $\mathcal{T}_2 \| \mathcal{T}_4$.

Case 2: $a \in \Sigma_2 \oplus \Sigma_4$. *Without loss of generality, we assume that* $a \in \Sigma_2$ *and* $a \notin \Sigma_4$. *Since* $\Sigma_2 \subseteq \Sigma_1$ *and* $\Sigma_4 \subseteq \Sigma_3$, *we have* $a \in \Sigma_1 \cap \Sigma_3$ *or* $a \in \Sigma_1 \setminus \Sigma_3$. *If* $a \in \Sigma_1 \cap \Sigma_3$ *then* $s_1 \xrightarrow{a?} s_1'$ *and* $s_3 \xrightarrow{a?} s_3'$ *according to the rule "SYNC-IN". By simulation definition we have that* $s_2 \Longrightarrow^{a?} s_2'$ *and* $s_4 \Longrightarrow^{\mathbf{0}} s_4'$ *for some* $s_2' \in S_2, s_4' \in S_4$ *such that* $(s_1', s_2') \in R_1$ *and* $(s_3', s_4') \in R_2$. *Hence* $(\langle s_1', s_3'\rangle, \langle s_2', s_4'\rangle) \in R$, *and from 3 of Lemma 2 there exists* $\langle s_2, s_4\rangle \Longrightarrow^{a?} \langle s_2', s_4'\rangle$ *in* $\mathcal{T}_2 \| \mathcal{T}_4$. *Otherwise* $a \in \Sigma_1 \setminus \Sigma_3$ *then* $s_1 \xrightarrow{a?} s_1'$ *and* $s_3 = s_3'$ *according to the rule "INDEP-L". From* $\mathcal{T}_1 \preceq \mathcal{T}_2$, *we have* $s_2 \Longrightarrow^{a?} s_2'$ *for some* $s_2' \in S_2$ *such that* $(s_1', s_2') \in R_1$. *Hence* $(\langle s_1', s_3\rangle, \langle s_2', s_4\rangle) \in R$. *From 3 of Lemma 2 there exists* $\langle s_2, s_4\rangle \Longrightarrow^{a?} \langle s_2', s_4\rangle$ *in* $\mathcal{T}_2 \| \mathcal{T}_4$.

Suppose $\langle s_1, s_3\rangle \xrightarrow{a!} \langle s_1', s_3'\rangle$ *for some* $a \in \Sigma_{2\|4}$. *There are also two cases:*

Case 1: $a \in \Sigma_2 \cap \Sigma_4$. *Since* $\Sigma_2 \subseteq \Sigma_1$ *and* $\Sigma_4 \subseteq \Sigma_3$, $a \in \Sigma_1 \cap \Sigma_3$. *According to the rule "SYNC-IO", we have that* $s_1 \xrightarrow{a?} s_1'$ *and* $s_3 \xrightarrow{a!} s_3'$, *or* $s_1 \xrightarrow{a!} s_1'$ *and* $s_3 \xrightarrow{a?} s_3'$. *By simulation definition we have* $s_2 \Longrightarrow^{a?} s_2'$, $s_4 \Longrightarrow^{a!} s_4'$, *or* $s_2 \Longrightarrow^{a!} s_2'$, $s_4 \Longrightarrow^{a?} s_4'$ *respectively such that* $(s_1', s_2') \in R_1$ *and* $(s_3', s_4') \in R_2$ *for some* $s_2' \in S_2, s_4' \in S_4$. *Hence* $(\langle s_1', s_3'\rangle, \langle s_2', s_4'\rangle) \in R$, *and from 2 of Lemma 2 there exists* $\langle s_2, s_4\rangle \Longrightarrow^{a!} \langle s_2', s_4'\rangle$ *in* $\mathcal{T}_2 \| \mathcal{T}_4$.

Case 2: $a \in \Sigma_2 \oplus \Sigma_4$. *Table 6 shows the possible transitions in* $\mathcal{T}_1$ *and* $\mathcal{T}_3$. *From the assumption that* $I_2 \cap O_3 \subseteq \Sigma_4$ *and* $O_1 \cap I_4 \subseteq \Sigma_2$, *there exist* $a \in \Sigma_4$ *in No. 1 and* $a \in \Sigma_2$ *in No. 5, which contradict their premises* $a \in \Sigma_2 \setminus \Sigma_4$ *and* $a \in \Sigma_4 \setminus \Sigma_2$ *respectively. Thus the cases of No. 1 and No. 5 will not exist. Consider the other cases in Table 6. By* $\mathcal{T}_1 \preceq \mathcal{T}_2$ *and* $\mathcal{T}_3 \preceq \mathcal{T}_4$ *we have that* (st_2, Tr_2, st_2') *in* $\rightarrow_2$ *and* (st_4, Tr_4, st_4') *in* $\rightarrow_4$ *for some* $st_2' \in S_2, st_4' \in S_4$ *such that* $(st_1', st_2') \in R_1$ *and* $(st_3', st_4') \in R_2$. *Hence* $(\langle st_1', st_3'\rangle, \langle st_2', st_4'\rangle) \in R$, *and from 3 of Lemma 2 there exists* $\langle st_2, st_4\rangle \Longrightarrow^{a!} \langle st_2', st_4'\rangle$ *in* $\mathcal{T}_2 \| \mathcal{T}_4$.

Suppose $\langle s_1, s_3\rangle \xrightarrow{a} \langle s_1', s_3'\rangle$ *for some* $a \in \Sigma_{1\|3} \setminus \Sigma_{2\|4}$. *Since* $\Sigma_{2\|4} = \Sigma_2 \cup \Sigma_4$, $a \in \Sigma_{1\|3} \setminus (\Sigma_2 \cup \Sigma_4)$. *Table 7 shows the possible transitions in* $\mathcal{T}_1$ *and* $\mathcal{T}_3$. *By* $\mathcal{T}_1 \preceq \mathcal{T}_2$ *and* $\mathcal{T}_3 \preceq \mathcal{T}_4$ *we have that* (st_2, Tr_2, st_2') *in* $\rightarrow_2$ *and* (st_4, Tr_4, st_4') *in* $\rightarrow_4$

Table 7 Transition Set 2

Premises	$\mathcal{T}_1$			$\mathcal{T}_2$			$\mathcal{T}_3$			$\mathcal{T}_4$		
	st_1	Tr_1	st_1'	st_2	Tr_2	st_2'	st_3	Tr_3	st_3'	st_4	Tr_4	st_4'
$a \in I_{1\|3}$	s_1	$\xrightarrow{a?}$	s_1'	s_2	$\overset{\mathbf{0}}{\Longrightarrow}$	s_2'	s_3	$\xrightarrow{\mathbf{0}}$	s_3	s_4	$\xrightarrow{\mathbf{0}}$	s_4
	s_1	$\xrightarrow{\mathbf{0}}$	s_1	s_2	$\xrightarrow{\mathbf{0}}$	s_2	s_3	$\xrightarrow{a?}$	s_3'	s_4	$\overset{\mathbf{0}}{\Longrightarrow}$	s_4'
	s_1	$\xrightarrow{a?}$	s_1'	s_2	$\overset{\mathbf{0}}{\Longrightarrow}$	s_2'	s_3	$\xrightarrow{a?}$	s_3'	s_4	$\overset{\mathbf{0}}{\Longrightarrow}$	s_4'
$a \in O_{1\|3}$	s_1	$\xrightarrow{a!}$	s_1'	s_2	$\overset{\mathbf{0}}{\Longrightarrow}$	s_2'	s_3	$\xrightarrow{\mathbf{0}}$	s_3	s_4	$\xrightarrow{\mathbf{0}}$	s_4
	s_1	$\xrightarrow{\mathbf{0}}$	s_1	s_2	$\xrightarrow{\mathbf{0}}$	s_2	s_3	$\xrightarrow{a!}$	s_3'	s_4	$\overset{\mathbf{0}}{\Longrightarrow}$	s_4'
	s_1	$\xrightarrow{a?}$	s_1'	s_2	$\overset{\mathbf{0}}{\Longrightarrow}$	s_2'	s_3	$\xrightarrow{a!}$	s_3'	s_4	$\overset{\mathbf{0}}{\Longrightarrow}$	s_4'
	s_1	$\xrightarrow{a!}$	s_1'	s_2	$\overset{\mathbf{0}}{\Longrightarrow}$	s_2'	s_3	$\xrightarrow{a?}$	s_3'	s_4	$\overset{\mathbf{0}}{\Longrightarrow}$	s_4'

Table 8 Transition Set 3

No.	$\mathcal{T}_1$			$\mathcal{T}_2$			$\mathcal{T}_3$			$\mathcal{T}_4$		
	st_1	Tr_1	st_1'	st_2	Tr_2	st_2'	st_3	Tr_3	st_3'	st_4	Tr_4	st_4'
1	s_1	$\xrightarrow{\tau}$	s_1'	s_2	$\overset{\mathbf{0}}{\Longrightarrow}$	s_2'	s_3	$\xrightarrow{\mathbf{0}}$	s_3	s_4	$\xrightarrow{\mathbf{0}}$	s_4
2	s_1	$\xrightarrow{\mathbf{0}}$	s_1	s_2	$\xrightarrow{\mathbf{0}}$	s_2	s_3	$\xrightarrow{\tau}$	s_3'	s_4	$\overset{\mathbf{0}}{\Longrightarrow}$	s_4'

for some $st_2' \in S_2, st_4' \in S_4$ such that $(st_1', st_2') \in R_1$ and $(st_3', st_4') \in R_2$. Hence $(\langle st_1', st_3'\rangle, \langle st_2', st_4'\rangle) \in R$, and from 4 of Lemma 2 there exists $\langle st_2, st_4\rangle \overset{\mathbf{0}}{\Longrightarrow} \langle st_2', st_4'\rangle$ in $\mathcal{T}_2 \| \mathcal{T}_4$.

Suppose $\langle s_1, s_3\rangle \xrightarrow{\tau} \langle s_1', s_3'\rangle$. Table 8 shows the possible transitions in $\mathcal{T}_1$ and $\mathcal{T}_3$. By $\mathcal{T}_1 \preceq \mathcal{T}_2$ and $\mathcal{T}_3 \preceq \mathcal{T}_4$ we have that (st_2, Tr_2, st_2') in $\rightarrow_2$ and (st_4, Tr_4, st_4') in $\rightarrow_4$ for some $st_2' \in S_2, st_4' \in S_4$ such that $(st_1', st_2') \in R_1$ and $(st_3', st_4') \in R_2$. Hence $(\langle st_1', st_3'\rangle, \langle st_2', st_4'\rangle) \in R$, and from 4 of Lemma 2 there exists $\langle st_2, st_4\rangle \overset{\mathbf{0}}{\Longrightarrow} \langle st_2', st_4'\rangle$ in $\mathcal{T}_2 \| \mathcal{T}_4$.

Suppose $\langle s_1, s_3\rangle \xrightarrow{\epsilon(d)} \langle s_1', s_3'\rangle$ and $d > 0$. Thus $s_1 \xrightarrow{\epsilon(d)} s_1'$ and $s_3 \xrightarrow{\epsilon(d)} s_3'$. By simulation definition we have that $s_2 \overset{\epsilon(d)}{\Longrightarrow} s_2'$ and $s_4 \overset{\epsilon(d)}{\Longrightarrow} s_4'$ for some $s_2' \in S_2, s_4' \in S_4$ such that $(s_1', s_2') \in R_1$ and $(s_3', s_4') \in R_2$. Hence $(\langle s_1', s_3'\rangle, \langle s_2', s_4'\rangle) \in R$, and from 4 of Lemma 2 there exists $\langle s_2, s_4\rangle \overset{\epsilon(d)}{\Longrightarrow} \langle s_2', s_4'\rangle$ in $\mathcal{T}_2 \| \mathcal{T}_4$. All the conditions hold and thus $\mathcal{T}_1 \| \mathcal{T}_3 \preceq \mathcal{T}_2 \| \mathcal{T}_4$.

C. Periodic task template

Figure 22 depicts the complete `PeriodicTask` template in the UPPAAL modeling framework.

For any periodic task in a partition P_i, its first release point is relative to the start of P_i's first partition window in the next partition period[5]. After starting from the initial location `WaitInitialOffset`, the model passes a series of locations that share a common name format "WaitXXX" where the suffix "XXX" denotes a specific delay like offset. For example, by defining the invariant `x<=pprd()-initialOffset()%pprd()` in which the function `pprd` returns the partition period, the location `WaitForPhase` waits until the start of the next partition period after the initial offset of the

task. Similarly, the task stays at location `WaitOffset` for the offset time. These locations help a periodic task determine its first release point when the task being ready to run joins the location `Ready`.

Thereafter, if the task is scheduled by `TaskScheduler` through the channel `sched`, it will start execution on the processor and move to the location `ReadOp`. For any task in the system, the sequential list $\mathcal{L}$ of its abstract instructions shown in section III.C is implemented by an array of structures `op`. By using an integer variable `pc` as a program counter, the task can fetch the next abstract instruction from `op[pc]` at the location `ReadOp`.

According to the command in the abstract instruction currently read from `op`, the task model performs a conditional branch and moves from the location `ReadOp` to one of the different locations that represent different operations. Therefore, the command set containing seven elements divides the rest of the template into seven parts, which have been described in section IV.D.2.

Funding Sources

This work was in part funded by Independent Research Fund Denmark under grant number DFF-7017-00348, Compositional Verification of Real-time MULTI-CORE SAFETY Critical Systems.

References

[1] Wolfig, R., and Jakovljevic, M., "Distributed IMA and DO-297: Architectural, communication and certification attributes," *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, IEEE, 2008, pp. 1–E.

[2] Wang, G., and Gu, Q., "Research on distributed integrated modular avionics system architecture design and implementation," *2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC)*, IEEE, 2013, pp. 7D6–1.

[3] Annighöfer, B., and Thielecke, F., *A Systems Architecting Framework for Distributed Integrated Modular Avionics*, Deutsche Gesellschaft für Luft-und Raumfahrt-Lilienthal-Oberth eV, 2014.

[4] AEEC, "Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network," ARINC specification 664P7-1, Aeronautical Radio Inc., Sep. 2009.

[5] AEEC, "Avionics Application Software Standard Interface: Part 1 - Required Services," ARINC specification 653P1-4, Aeronautical Radio Inc., Aug. 2015.

[6] Dodd, R., "Coloured petri net modelling of a generic avionics mission computer," Tech. rep., DTIC Document, 2006.

[7] Bucci, G., Fedeli, A., Sassoli, L., and Vicario, E., "Modeling flexible real time systems with preemptive time Petri nets," *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, IEEE, 2003, pp. 279–286.

[8] Alur, R., Courcoubetis, C., Henzinger, T. A., and Ho, P.-H., "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," *Hybrid systems*, Springer, 1993, pp. 209–229.

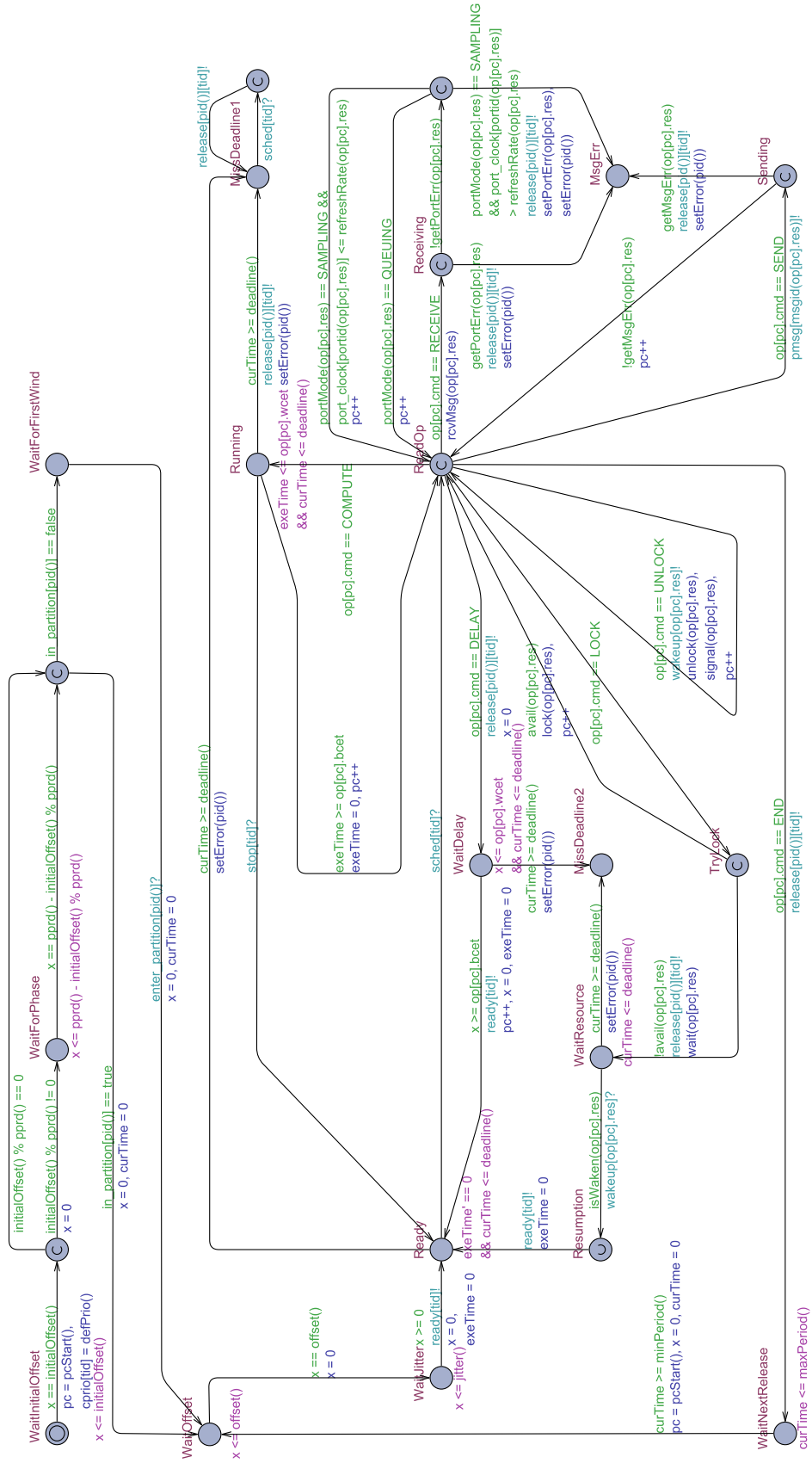


Fig. 22 Periodic task template

[9] Alur, R., and Dill, D. L., "A theory of timed automata," *Theoretical computer science*, Vol. 126, No. 2, 1994, pp. 183–235.

[10] Cassez, F., and Larsen, K., "The impressive power of stopwatches," *International Conference on Concurrency Theory*, Springer, 2000, pp. 138–152.

[11] Carnevali, L., Pinzuti, A., and Vicario, E., "Compositional verification for hierarchical scheduling of real-time systems," *IEEE Transactions on Software Engineering*, Vol. 39, No. 5, 2013, pp. 638–657.

[12] Sun, Y., Lipari, G., Soulat, R., Fribourg, L., and Markey, N., "Component-based analysis of hierarchical scheduling using linear hybrid automata," *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, 2014, pp. 1–10.

[13] Boudjadar, J., Larsen, K. G., Kim, J. H., and Nyman, U., "Compositional schedulability analysis of an avionics system using UPPAAL," *International Conference on Advanced Aspects of Software Engineering*, 2014.

[14] David, A., Larsen, K. G., Legay, A., Mikučionis, M., and Poulsen, D. B., "Uppaal SMC tutorial," *STTT*, Vol. 17, No. 4, 2015, pp. 397–415. doi:10.1007/s10009-014-0361-y.

[15] Han, P., Zhai, Z., Nielsen, B., and Nyman, U., "A Modeling Framework for Schedulability Analysis of Distributed Avionics Systems," *arXiv preprint arXiv:1803.11050*, 2018.

[16] Han, P., Zhai, Z., Nielsen, B., and Nyman, U., "A Compositional Approach for Schedulability Analysis of Distributed Avionics Systems," *arXiv preprint arXiv:1807.11570*, 2018.

[17] Fuchsen, R., "IMA NextGen: A new technology for the Scarlett program," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 25, No. 10, 2010, pp. 10–16.

[18] Mok, A. K., Feng, X., and Chen, D., "Resource partition for real-time systems," *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*, IEEE, 2001, pp. 75–84.

[19] Shin, I., and Lee, I., "Periodic resource model for compositional real-time guarantees," *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, IEEE, 2003, pp. 2–13.

[20] Shin, I., and Lee, I., "Compositional real-time scheduling framework," *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, IEEE, 2004, pp. 57–67.

[21] Easwaran, A., Anand, M., and Lee, I., "Compositional analysis framework using EDP resource models," *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, IEEE, 2007, pp. 129–138.

[22] Easwaran, A., Lee, I., Sokolsky, O., and Vestal, S., "A compositional scheduling framework for digital avionics systems," *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, 2009, pp. 371–380.

[23] Kim, J.-E., Abdelzaher, T., and Sha, L., "Schedulability bound for integrated modular avionics partitions," *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, EDA Consortium, 2015, pp. 37–42.

[24] Carnevali, L., Lipari, G., Pinzuti, A., and Vicario, E., "A formal approach to design and verification of two-level Hierarchical Scheduling systems," *International Conference on Reliable Software Technologies*, Springer, 2011, pp. 118–131.

[25] Sun, Y., and Lipari, G., "A pre-order relation for exact schedulability test of sporadic tasks on multiprocessor Global Fixed-Priority scheduling," *Real-Time Systems*, Vol. 52, No. 3, 2016, pp. 323–355.

[26] Åsberg, M., Pettersson, P., and Nolte, T., "Modelling, verification and synthesis of two-tier hierarchical fixed-priority preemptive scheduling," *2011 23rd Euromicro Conference on Real-Time Systems*, IEEE, 2011, pp. 172–181.

[27] Fribourg, L., Soulat, R., Lesens, D., and Moro, P., "Robustness analysis for scheduling problems using the inverse method," *2012 19th International Symposium on Temporal Representation and Reasoning*, IEEE, 2012, pp. 73–80.

[28] Cicirelli, F., Furfaro, A., Nigro, L., and Pupo, F., "Development of a schedulability analysis framework based on PTPN and Uppaal with stopwatches," *Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, IEEE Computer Society, 2012, pp. 57–64.

[29] Boudjadar, J., David, A., Kim, J. H., Larsen, K. G., Nyman, U., and Skou, A., "Schedulability and energy efficiency for multi-core hierarchical scheduling systems," *Embedded Real Time Systems and Software*, 2014, pp. 1–4.

[30] Boudjadar, J., Kim, J. H., and Nadjm-Tehrani, S., "Performance-aware scheduling of multicore time-critical systems," *Formal Methods and Models for System Design (MEMOCODE), 2016 ACM/IEEE International Conference on*, IEEE, 2016, pp. 105–114.

[31] Scharbarg, J.-L., and Fraboul, C., "Simulation for end-to-end delays distribution on a switched ethernet," *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, IEEE, 2007, pp. 1092–1099.

[32] Safwat, N. E.-D., Zekry, A., and Abouelatta, M., "Avionics Full-duplex switched Ethernet (AFDX): Modeling and simulation," *Radio Science Conference (NRSC), 2015 32nd National*, IEEE, 2015, pp. 286–296.

[33] Rivas, J. M., Gutiérrez, J. J., Palencia, J. C., et al., "Schedulability analysis and optimization of heterogeneous EDF and FP distributed real-time systems," *2011 23rd Euromicro Conference on Real-Time Systems*, IEEE, 2011, pp. 195–204.

[34] Gutiérrez, J. J., Palencia, J. C., and Harbour, M. G., "Holistic schedulability analysis for multipacket messages in AFDX networks," *Real-Time Systems*, Vol. 50, No. 2, 2014.

[35] Le Boudec, J.-Y., and Thiran, P., *Network calculus: a theory of deterministic queuing systems for the internet*, Vol. 2050, Springer Science & Business Media, 2001.

[36] Scharbarg, J.-L., Ridouard, F., and Fraboul, C., "A probabilistic analysis of end-to-end delays on an AFDX avionic network," *IEEE transactions on industrial informatics*, Vol. 5, No. 1, 2009, pp. 38–49.

[37] Bauer, H., Scharbarg, J.-L., and Fraboul, C., "Improving the worst-case delay analysis of an AFDX network using an optimized trajectory approach," *IEEE Transactions on Industrial informatics*, Vol. 6, No. 4, 2010, pp. 521–533.

[38] Kemayo, G., Ridouard, F., Bauer, H., and Richard, P., "A Forward end-to-end delays Analysis for packet switched networks," *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, ACM, 2014, p. 65.

[39] Kemayo, G., Benammar, N., Ridouard, F., Bauer, H., and Richard, P., "Improving AFDX end-to-end delays analysis," *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, IEEE, 2015, pp. 1–8.

[40] Adnan, M., Scharbarg, J.-L., Ermont, J., and Fraboul, C., "Model for worst case delay analysis of an AFDX network using timed automata," *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, IEEE, 2010, pp. 1–4.

[41] Scharbarg, J.-L., and Fraboul, C., *Methods and tools for the temporal analysis of avionic networks*, INTECH Open Access Publisher, 2010.

[42] Adnan, M., Scharbarg, J.-L., Ermont, J., and Fraboul, C., "An improved timed automata approach for computing exact worst-case delays of AFDX sporadic flows," *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, IEEE, 2012, pp. 1–8.

[43] Grumberg, O., and Long, D., "Model checking and modular verification," *Toplas*, Vol. 16, No. 3, 1994, pp. 843–871.

[44] FAA, "CAST-32A Multi-core Processors," *CAST Position Paper*, 2016.

[45] Jean, X., Gatti, M., Berthon, G., and Fumey, M., "MULCORS-Use of Multicore Processors in airborne systems," *EASA, Tech. Rep.*, 2012.

[46] Huyck, P., "ARINC 653 and multi-core microprocessors—Considerations and potential impacts," *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, IEEE, 2012, pp. 1–16.

[47] MATTERS, W. I., and RIVER, I. R. O. W., "Certification of Avionics Applications on Multi-core Processors: Opportunities and Challenges," 2018.

[48] Parkinson, P. J., "Applying MILS to multicore avionics systems," *Proc. Int. Workshop Mils, Archit. Assurance Secur. Syst.(HIPEAC)*, 2016, pp. 1–9.

[49] Ittershagen, P., Hartmann, P. A., Grüttner, K., and Rettberg, A., "Hierarchical real-time scheduling in the multi-core era - An overview," *16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*, 2013, pp. 1–10. doi:10.1109/ISORC.2013.6913241.

[50] Sha, L., Rajkumar, R., and Lehoczky, J. P., "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on computers*, Vol. 39, No. 9, 1990, pp. 1175–1185.

[51] David, A., Larsen, K. G., Legay, A., Nyman, U., and Wasowski, A., "Timed I/O automata: a complete specification theory for real-time systems," *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, ACM, 2010, pp. 91–100.

[52] Berezin, S., Campos, S., and Clarke, E. M., "Compositional reasoning in model checking," *International Symposium on Compositionality*, Springer, 1997, pp. 81–102.

[53] Jensen, H. E., Larsen, K. G., and Skou, A., "Scaling up UPPAAL," *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, Springer, 2000, pp. 19–30.

[54] Jensen, H., "Abstraction-based verification of distributed systems," Ph.D. thesis, Aalborg university, 1999.

[55] Reineke, J., Wachter, B., Thesing, S., Wilhelm, R., Polian, I., Eisinger, J., and Becker, B., "A definition and classification of timing anomalies," *OASIcs-OpenAccess Series in Informatics*, Vol. 4, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.